

# Bridging Rapid Simulation and Intelligent Reasoning: Geometry-Aware Neural Operators and Multi-Agent LLM Frameworks for Engineering Design

By

Varun Kumar

B.Tech., Birla Institute of Technology and Science, Ranchi, India, 2011

M.Sc., Clemson University, SC, USA, 2016

A dissertation submitted in partial fulfillment of the requirements for the  
degree of Doctor of Philosophy in the School of Engineering at Brown  
University

PROVIDENCE, RHODE ISLAND

May 2026

© Copyright 2026 (Varun Kumar)



This dissertation by Varun Kumar is accepted in its present form by the School of Engineering as satisfying the dissertation requirement for the degree of Doctor of Philosophy.

Date \_\_\_\_\_  
George Em Karniadakis, Advisor

Recommended to the Graduate Council

Date \_\_\_\_\_  
Miguel Bessa, Reader

Date \_\_\_\_\_  
Mauro Rodriguez, Reader

Approved by the Graduate Council

Date \_\_\_\_\_  
David P. Lindstrom, Dean of the Graduate School

# Curriculum Vitae

## Education

**2022–2026 Ph.D., (Fluids and Thermal Sciences)**

Brown University (Providence, RI, USA)

**2014–2016**

**M.Sc., (Mechanical Engineering)**

Clemson University (Clemson, SC, USA)

**2007–2011**

**B.Tech, (Mechanical Engineering)**

Birla Institute of Technology and Science (Ranchi, India)

## Publications

### Journal Papers

Kamaljyoti Nath, **Varun Kumar**, Daniel J. Smith, and George Em Karniadakis. A digital twin for diesel engines: Operator-infused physics-informed neural networks with transfer learning for engine health monitoring. *Engineering Applications of Artificial Intelligence*, volume 170, page 114052, 2026.

Ahmad Peyvan, **Varun Kumar**, and George Em Karniadakis. Fusion-deeponet: A dataefficient neural operator for geometry-dependent hypersonic and supersonic flows. *Journal of Computational Physics*, page 114432. Elsevier, 2025.

**Varun Kumar**, Somdatta Goswami, Katiana Kontolati, Michael D Shields, and George Em Karniadakis. Synergistic learning with multi-task deeponet for efficient pde problem solving. *Neural Networks*, volume 184, page 107113. Elsevier, 2025

**Varun Kumar**, Somdatta Goswami, Daniel Smith, and George Em Karniadakis. Real-time prediction of gas flow dynamics in diesel engines using a deep neural operator framework: V. kumar et al. *Applied Intelligence*, volume 54, pages 14–34. Springer, 2024.

**Varun Kumar**, Leonard Gleyzer, Adar Kahana, Khemraj Shukla, and George Em Karniadakis. MyCrunchGPT: A llm assisted framework for scientific machine learning. *Journal of Machine Learning for Modeling and Computing*, volume 4. Begel House Inc., 2023.

## Conference Proceedings

**Varun Kumar** and Gregory Mocko. Similarity of engineering design problems to enable reuse in design research experiments. In International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, volume 50190, page V007T06A042. American Society of Mechanical Engineers, 2016.

**Varun Kumar**, Gregory Mocko. Design problems as potential moderators in creativity studies. In DS 86: Proceedings of The Fourth International Conference on Design Creativity, Georgia Institute of Technology, Atlanta, GA, USA, 2016.

## PrePrints/Under Review

**Varun Kumar** and George Em Karniadakis. Toward autonomous engineering design: A knowledge-guided multi-agent framework (under review). arXiv preprint arXiv:2511.03179, 2025.

**Varun Kumar**, Jing Bi, Cyril Ngo Ngoc, Victor Oancea, and George Em Karniadakis. Learning nonlinear responses in pet bottle buckling with a hybrid deepoNet-transolver framework (under review). arXiv preprint arXiv:2509.13520, 2025.

**Varun Kumar** and George Em Karniadakis. Agentic Risk-Aware Set-Based Engineering Design. arXiv preprint arXiv:2604.16687, 2026.

## Work Experience

**2017-2022**

**Systems Engineer**

Fujifilm Dimatix Inc., Lebanon, NH, USA

**2011-2014**

**Engine Design and Development Engineer**

Mahinda and Mahindra Ltd., Chennai, India

## Invited Talks

**2025, Presented multi-agent framework proposed in ‘Toward Autonomous Engineering Design: A Knowledge-Guided Multi-Agent Framework’**

*Atomic Machines LLC*

**2025, Presented multi-task framework proposed in ‘Synergistic learning with multi-task DeepONet for efficient PDE problem solving’**

*USNCCM18, Chicago, IL, USA*

**2024, Presented ‘Data-Driven Domain Decomposition: Predicting Responses in Diverse 3D Geometries using Neural Operator-Based Framework’**  
*ECCOMAS*, Lisbon, Portugal

**2016, Presented design problem elements proposed in ‘Similarity of engineering design problems to enable reuse in design research experiments’**  
*ASME IDETC*, Charlotte, NC, USA

## Teaching experience

**Teaching assistant, ME4400: Materials for Aggressive Environments**  
*Clemson University*

**Student mentor, Senior Capstone Design projects**  
*Clemson University*

## Awards and Fellowships

**USNCCM Travel Grant, Travel grant to attend USNCCM’18 conference**  
*Chicago, USA*

**Brown University, Travel grant to attend ECCOMAS 2024 conference**  
*Lisbon, Portugal*

# Acknowledgments

I would like to express my gratitude to everyone who has supported and motivated me through this journey. This work would not have been possible without the collective encouragement and expertise of many individuals.

First and foremost, I want to thank my advisor, Prof. George Karniadakis, for his guidance, patience, and mentorship. His critical assessment of my ideas and methods pushed me to think more deeply about my research. His vision and ideas were instrumental in shaping this research. I am also grateful to my committee members, Dr. Miguel Bessa and Dr. Mauro Rodriguez, for their valuable feedback and for challenging me to think critically about my findings and methods.

A special thanks to my collaborators at Brown University, Dr. Somdatta Goswami, Dr. Ahmad Peyvan, and Dr. Kamaljoti Nath for their stimulating discussions. I also thank my external collaborators, Jing Bi, Victor Oancea, Cyril Ngoc at Simulia and Daniel J. Smith at Cummins for giving me the opportunity to work on some exciting problems from their respective domains. I would also like to thank my funding agencies, namely DARPA, MURI, ONR Vannevar Bush Faculty Fellowship, and US-DOE SEA-CROGS for their financial support to conduct this research.

To my parents and in-laws, thank you for the lifelong encouragement and for believing in my abilities to succeed in my endeavors. Most importantly, I want to thank my wife, Ankita for being my rock, for her endless patience, and celebrating every small victory along the way. This achievement was impossible without you by my side.

# List of Publications and Contribution Statement

The following list of publications were used with permission from respective journals in preparation of this thesis:

1. **Kumar, V.**, Goswami, S., Kontolati, K., Shields, M.D., Karniadakis, G. “Synergistic learning with multi-task DeepONet for efficient PDE problem solving” *Neural Networks*, Volume 184, 2025. doi: <https://doi.org/10.1016/j.neunet.2024.107113>. (Basis for Chapter 2).
2. Peyvan, A., **Kumar, V.**, Karniadakis, G. “Fusion-DeepONet: A data-efficient neural operator for geometry-dependent hypersonic and supersonic flows” *Journal of Computational Physics*, Volume 544, 2026. doi: <https://doi.org/10.1016/j.jcp.2025.114432> (Basis for Chapter 3).

**Contributions:** My contributions in this work include:

- Benchmarking existing SOTA geometry-aware deep learning methods including U-Nets, MeshGraphNet, FNO against the proposed Fusion-DeepONet framework for semi-ellipse and nozzle problems,
  - Literature review for existing works in geometry-aware deep learning models,
  - Developing U-Net framework for learning the map between geometric mask and flow fields,
  - Modifying MeshGraphNet and FNO frameworks to accommodate geometric parameters as inputs, for comparable results with Fusion DeepONet,
  - Writing background information and method details for MeshGraphNets, FNO, and U-Net.
3. **Kumar, V.**, Gleyzer, L., Kahana, A., Shukla, K., Karniadakis, G. “Mycrunchgpt: A llm assisted framework for scientific machine learning” *Journal of Machine Learning for Modeling and Computing*, Volume 4, 2023. doi: [10.1615/JMachLearnModelComput.2023049518](https://doi.org/10.1615/JMachLearnModelComput.2023049518). (Basis for Chapter 6).

**Contributions:** My contributions in this work include development of LLM framework for design and optimization of airfoils using a DeepONet surrogate.

The following works were under review/submitted to Pre-Print at the time of writing this thesis

1. **Kumar, V.**, Bi, J., Ngoc, C.N., Oancea, V., Karniadakis, G. “Learning Nonlinear Responses in PET Bottle Buckling with a Hybrid DeepONet-Transolver Framework” doi: <https://doi.org/10.48550/arXiv.2509.13520>, 2025. (Basis for Chapter 4, under review).
2. **Kumar, V.**, Karniadakis, G. “Toward Autonomous Engineering Design: A Knowledge-Guided Multi-Agent Framework” doi: <https://doi.org/10.48550/arXiv.2511.03179>, 2025. (Basis for Chapter 7, under review).
3. **Kumar, V.**, Karniadakis, G. “Agentic Risk-Aware Set-Based Engineering Design” doi: <https://doi.org/10.48550/arXiv.2604.16687>, 2026. (Basis for Chapter 8).

# Contents

|                                                                                                                   |             |
|-------------------------------------------------------------------------------------------------------------------|-------------|
| <b>Acknowledgments</b>                                                                                            | <b>vii</b>  |
| <b>List of Publications and Contribution Statement</b>                                                            | <b>viii</b> |
| <b>1 Introduction to Geometry-Aware Neural Operators</b>                                                          | <b>2</b>    |
| <b>2 Synergistic Learning with Multi-Task DeepONet for Efficient PDE Problem Solving</b>                          | <b>6</b>    |
| 2.1 Introduction . . . . .                                                                                        | 6           |
| 2.2 Multi-task learning in neural operators . . . . .                                                             | 10          |
| 2.2.1 Deep operator network . . . . .                                                                             | 10          |
| 2.2.2 Multi-task deep operator network (MT-DeepONet) . . . . .                                                    | 12          |
| 2.3 Numerical examples . . . . .                                                                                  | 17          |
| 2.3.1 Fisher Equations . . . . .                                                                                  | 17          |
| 2.3.2 Darcy Flow in 2D geometries . . . . .                                                                       | 20          |
| 2.3.3 Heat transfer through multiple 3D geometries . . . . .                                                      | 29          |
| 2.4 Limitations and future work . . . . .                                                                         | 33          |
| 2.5 Summary . . . . .                                                                                             | 35          |
| <b>3 Fusion-DeepONet: A Data-Efficient Neural Operator for Geometry-Dependent Hypersonic and Supersonic Flows</b> | <b>38</b>   |
| 3.1 Introduction . . . . .                                                                                        | 38          |



|          |                                                                                                            |           |
|----------|------------------------------------------------------------------------------------------------------------|-----------|
| 3.2      | Governing Equations . . . . .                                                                              | 43        |
| 3.3      | Data Generation . . . . .                                                                                  | 45        |
| 3.3.1    | Inviscid hypersonic flow around semi-ellipses . . . . .                                                    | 46        |
| 3.3.2    | Hypersonic viscous flow around Reentry capsule . . . . .                                                   | 48        |
| 3.3.3    | Viscous flow inside convergent-divergent nozzle . . . . .                                                  | 51        |
| 3.4      | Methodology . . . . .                                                                                      | 53        |
| 3.4.1    | Fusion-DeepONet . . . . .                                                                                  | 54        |
| 3.4.2    | Derivative enhanced loss function . . . . .                                                                | 59        |
| 3.5      | Results . . . . .                                                                                          | 61        |
| 3.5.1    | Inviscid hypersonic flow around semi-ellipses . . . . .                                                    | 62        |
| 3.5.2    | Hypersonic viscous flow around reentry capsule . . . . .                                                   | 73        |
| 3.5.3    | Viscous flow inside convergent-divergent nozzle . . . . .                                                  | 80        |
| 3.6      | Summary . . . . .                                                                                          | 82        |
| <b>4</b> | <b>Learning Nonlinear Responses in PET Bottle Buckling with a Hybrid<br/>DeepONet–Transolver Framework</b> | <b>85</b> |
| 4.1      | Introduction . . . . .                                                                                     | 85        |
| 4.2      | Background and related works . . . . .                                                                     | 87        |
| 4.3      | Problem setting . . . . .                                                                                  | 91        |
| 4.3.1    | Geometry selection . . . . .                                                                               | 92        |
| 4.3.2    | Simulation details . . . . .                                                                               | 94        |
| 4.4      | Method . . . . .                                                                                           | 95        |
| 4.4.1    | Transolver . . . . .                                                                                       | 95        |
| 4.4.2    | DeepONet . . . . .                                                                                         | 98        |
| 4.4.3    | Hybrid DeepONet-Transolver framework . . . . .                                                             | 99        |
| 4.5      | Experiments . . . . .                                                                                      | 100       |
| 4.5.1    | Implementation details . . . . .                                                                           | 100       |
| 4.5.2    | Main results . . . . .                                                                                     | 103       |

|          |                                                                                       |            |
|----------|---------------------------------------------------------------------------------------|------------|
| 4.6      | Limitations and Challenges . . . . .                                                  | 108        |
| 4.7      | Summary . . . . .                                                                     | 109        |
| <b>5</b> | <b>Introduction to LLM-assisted Multi-Agent Frameworks for Engineering Designs</b>    | <b>112</b> |
| <b>6</b> | <b>MyCrunchGPT: A LLM assisted framework for scientific machine learning</b>          | <b>115</b> |
| 6.1      | Introduction . . . . .                                                                | 115        |
| 6.1.1    | Objective and contributions . . . . .                                                 | 117        |
| 6.2      | Scientific Machine Learning (SciML) . . . . .                                         | 118        |
| 6.2.1    | Physics-Informed Neural Networks . . . . .                                            | 119        |
| 6.2.2    | Deep Operator Network (DeepONet) . . . . .                                            | 120        |
| 6.3      | MyCrunchGPT . . . . .                                                                 | 121        |
| 6.3.1    | MyCrunchGPT for Operator-based Design . . . . .                                       | 121        |
| 6.3.2    | MyCrunchGPT for PINN-based simulations . . . . .                                      | 123        |
| 6.4      | Exemplar for MyCrunchGPT assisted Operator-based Design framework                     | 123        |
| 6.4.1    | Problem subset 1: Generating new airfoil designs . . . . .                            | 124        |
| 6.4.2    | Problem subset 2: Analyzing flow fields around new designs . . .                      | 125        |
| 6.4.3    | Problem subset3: Generate optimized airfoil geometry . . . . .                        | 126        |
| 6.4.4    | Use-case analysis . . . . .                                                           | 128        |
| 6.4.5    | Workflow for DeepONet-based airfoil design . . . . .                                  | 129        |
| 6.4.6    | MyCrunchGPT assisted result validation and comparison . . . . .                       | 137        |
| 6.5      | Exemplar for AI-assisted PINN-based analysis . . . . .                                | 139        |
| 6.6      | Limitations . . . . .                                                                 | 141        |
| 6.7      | Summary . . . . .                                                                     | 142        |
| <b>7</b> | <b>Toward Autonomous Engineering Design: A Knowledge-Guided Multi-Agent Framework</b> | <b>148</b> |

|          |                                                                          |            |
|----------|--------------------------------------------------------------------------|------------|
| 7.1      | Introduction . . . . .                                                   | 148        |
| 7.2      | Background and Related Works . . . . .                                   | 151        |
| 7.3      | Problem Setting . . . . .                                                | 155        |
| 7.4      | Methodology . . . . .                                                    | 157        |
| 7.4.1    | Knowledge Graphs for Engineering Agents . . . . .                        | 157        |
| 7.4.2    | Graph Ontologist Agent . . . . .                                         | 159        |
| 7.4.3    | Systems Engineer Agent . . . . .                                         | 161        |
| 7.4.4    | Design Engineer Agent . . . . .                                          | 164        |
| 7.4.5    | Manager (Human User) . . . . .                                           | 167        |
| 7.4.6    | Workflow . . . . .                                                       | 168        |
| 7.5      | Experiments . . . . .                                                    | 171        |
| 7.6      | Limitations and Future Works . . . . .                                   | 181        |
| 7.7      | Summary . . . . .                                                        | 182        |
| <b>8</b> | <b>Agentic Risk-Aware Set-Based Engineering Design</b>                   | <b>184</b> |
| 8.1      | Introduction . . . . .                                                   | 184        |
| 8.2      | Problem statement . . . . .                                              | 188        |
| 8.3      | Agentic Design as Sequential Decision-Making Under Uncertainty . . . . . | 189        |
| 8.4      | Methodology . . . . .                                                    | 192        |
| 8.4.1    | Agent: Coding Assistant . . . . .                                        | 193        |
| 8.4.2    | Agent: Design Engineer . . . . .                                         | 193        |
| 8.4.3    | Agent: Systems Engineer . . . . .                                        | 195        |
| 8.4.4    | Agent: Analyst . . . . .                                                 | 195        |
| 8.4.5    | Tools . . . . .                                                          | 196        |
| 8.4.6    | Filters . . . . .                                                        | 201        |
| 8.5      | Workflow and Experiments . . . . .                                       | 204        |
| 8.5.1    | Stage 0: Tool development . . . . .                                      | 204        |
| 8.5.2    | Stage 1: User request . . . . .                                          | 205        |

|          |                                                                    |            |
|----------|--------------------------------------------------------------------|------------|
| 8.5.3    | Stage 2: Design generation phase . . . . .                         | 206        |
| 8.5.4    | Stage 3: Sensitivity analysis . . . . .                            | 206        |
| 8.5.5    | Stage 4: Design revision and risk-based filtering . . . . .        | 209        |
| 8.5.6    | Stage 5: Design Revision and evaluation of Pressure distribution . | 211        |
| 8.5.7    | Stage 6: Automated design review and filtering . . . . .           | 212        |
| 8.5.8    | Iterative Human-in-the-Loop Design Review and Refinement . . .     | 215        |
| 8.5.9    | Stage 7: CFD simulation with OpenFoam . . . . .                    | 218        |
| 8.6      | Limitations . . . . .                                              | 223        |
| 8.7      | Summary . . . . .                                                  | 224        |
| <b>9</b> | <b>Summary and Future Work</b>                                     | <b>226</b> |
| 9.1      | Summary . . . . .                                                  | 226        |
| 9.2      | Future Work . . . . .                                              | 228        |
|          | <b>Appendices</b>                                                  | <b>230</b> |
| A1       | Appendices for Chapter 2 . . . . .                                 | 231        |
| A1.1     | Empirical results for two-branch framework (MIONet) . . . . .      | 231        |
| A1.2     | Data Generation . . . . .                                          | 231        |
| A1.3     | Data pre-processing . . . . .                                      | 234        |
| A1.4     | Network Architecture . . . . .                                     | 235        |
| A2       | Appendices for Chapter 3 . . . . .                                 | 236        |
| A2.1     | Datasets . . . . .                                                 | 236        |
| A2.2     | High pressure ratio Sod problem (LeBlanc problem) . . . . .        | 237        |
| A2.3     | Vanilla- and POD-DeepONet Hyperparameter Tuning . . . . .          | 238        |
| A2.4     | Neural Operators . . . . .                                         | 239        |
| A3       | Appenedix for Chapter 7 . . . . .                                  | 248        |
| A3.1     | Knowledge Graph information retrieval . . . . .                    | 248        |
| A3.2     | Reasoning behind Systems Engineer agent’s design review process    | 251        |
| A4       | Appendices for Chapter 8 . . . . .                                 | 255        |

|                   |                                                                 |            |
|-------------------|-----------------------------------------------------------------|------------|
| A4.1              | Related works . . . . .                                         | 255        |
| A4.2              | Bayesian surrogates . . . . .                                   | 257        |
| A4.3              | DeepONet surrogate . . . . .                                    | 261        |
| A4.4              | Stage 3: Sensitivity Analysis . . . . .                         | 264        |
| A4.5              | Stage 6: Automated Design review (additional information) . . . | 265        |
| A4.6              | Stage 7: OpenFoam CFD additional results . . . . .              | 268        |
| <b>References</b> |                                                                 | <b>308</b> |

# List of Figures

|     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |    |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 2.1 | A schematic representation of the operator learning benchmarks and MTL scenarios considered in this study. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | 10 |
| 2.2 | Schematic of the MT-DeepONet designed to learn a family of parametric PDEs (Fisher equations) defined by different forcing functions, $F(u)$ . The parametric representation of this equation family, along with random initial condition fields, is input to the branch network. Spatio-temporal points are input to the trunk network. The multi-task operator network learns to predict the solution field, $u$ across this parameterized family of equations and random initial solutions concurrently. . . . .                                                                                                                                                                                                                                                                                         | 14 |
| 2.3 | Schematic showing the binary masking function for a triangular geometry. A uniform grid $100 \times 100$ is sampled in $\Omega \in [0, 1] \times [0, 1]$ and used for generating the basis function in the trunk network. The binary mask is constructed by delineating the boundaries of the domain. Grid points within the domain boundary are denoted by 1, while those outside are denoted by 0. The binary mask is applied to the solution from the operator network, enforcing the solution outside the geometry to be 0, thereby aiding with network convergence. Additionally, the binary mask also establishes sample uniqueness by enforcing the geometric constraint on the output space of MT-DeepONet thereby ensuring a bijective mapping between input and output functional spaces. . . . . | 15 |

|     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |    |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 2.4 | Comparison between the reference solution and prediction obtained using the proposed multi-task operator network for representative test cases. The plot shows the operator network predictions against reference solutions for different initial conditions and forcing functions $F(u)$ . The results demonstrate good overall accuracy across different initial conditions and forcing functions. . . . .                                                                                                                                                                                                                                                                                                                                          | 20 |
| 2.5 | Different geometries considered as tasks for MT-DeepONet for the Darcy flow problem. The source MT-DeepONet is trained using various combinations of the geometries S1 - S7 while the target domain geometries considered are T1 - T3. The boundary indicated in red denotes the Dirichlet boundaries where the solution $h(x) = 0$ . The first objective of MT-DeepONet is to learn the hydraulic pressure heads across a combination of source geometries given a parametric family of spatially varying conductivity fields. The second objective is to transfer the knowledge of source MT-DeepONet to different geometries in the target domain using the transfer learning approach proposed in [1] to reduce overall computation time. . . . . | 22 |
| 2.6 | Representative prediction results for source model trained on geometries <b>S1</b> + <b>S2</b> + <b>S3</b> , concurrently. The MT-DeepONet source model captures the overall features of the solution space across all three geometries reasonably well, making it suitable for use in the transfer learning process. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                         | 24 |

|     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |    |
|-----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 2.7 | Comparative analysis between reference solution and prediction generated by the target model <b>T1</b> trained with 50 samples. Source model <b>S1 + S2</b> results in lower error values than source model <i>S1</i> for this geometry. Other source models using <b>S3</b> in training samples lead to higher errors due to negative interference. Note the artifacts that emerge due to negative interference from the triangular source geometry for source model combinations ( <b>S1 + S2 + S3</b> ) and ( <b>S1 + S3</b> ). Overall, the target model is capable of capturing the pressure distribution for new geometries relatively well with the transfer learning methodology when appropriate geometries are chosen in the source model. . . . . | 26 |
| 2.8 | Comparative analysis between reference solution and prediction generated by the target model <b>T2</b> fine-tuned with 50 samples. The absolute error plot unveils certain regions with higher error accumulation, especially surrounding the cutout in a square geometric domain ( <b>T2</b> ). Target geometry <b>T2</b> represents an extrapolated case since none of the source models have an internal Dirichlet boundary with zero pressure head enforced. . . . .                                                                                                                                                                                                                                                                                     | 27 |
| 2.9 | A comparison between the reference solution and the prediction from our multi-task operator network for the transfer learning model, trained with 50 samples on geometry combinations of <b>S3-S7</b> . The I-section geometry used as the target is significantly different from the geometries used in the source model. Utilizing a source model trained on multiple geometries for transfer learning results in a marginally lower prediction error which may not be significant. . . . .                                                                                                                                                                                                                                                                | 29 |



|      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |    |
|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 2.10 | (a) View of a 3D plate showing the parametric design features, where $\mathbf{n}$ represents the number of holes and $\mathbf{d}$ denotes the location of the holes relative to the central axis of the protrusion face. Multiple geometries are generated by varying $\mathbf{n}$ and $\mathbf{d}$ within their design ranges. (b) depicts the boundary conditions used for solving the heat transfer equation. The top protrusion and the side wall have a convective heat flux boundary condition, while the holes act as individual heat sources. The solution field consists of the temperature distribution across the 3D plate domain. . . . .                                         | 31 |
| 2.11 | (a) Representative 3D plate geometry configurations generated by changing the two design parameters: $\mathbf{n}$ and $\mathbf{d}$ . (b) Variations in temperature fields for different geometric configurations and number of heating sources. The multi-task objective is to learn the mapping from geometric parameters to temperature fields for different geometric configurations. . . . .                                                                                                                                                                                                                                                                                              | 32 |
| 2.12 | Comparison of convergence rates between a network trained with the loss function as a product of the solution operator and the masking function (left) and one trained without a masking operation in the loss function (right). The application of the mask to the solution output demonstrates better error convergence and a reduction in generalization error. . . . .                                                                                                                                                                                                                                                                                                                    | 33 |
| 2.13 | Four representative geometries with different numbers and locations of the holes to compare the accuracy of the MT-DeepONet solution (without the masking operation) with the reference solution. The results revealed higher errors across the plate compared to the masked solution (see Figure 2.14). Applying the binary mask imposes an additional constraint that assists the network in learning solutions across varying geometries, particularly when inferring new geometries not seen during training. The masking operation provides crucial geometric information, enhancing the network's ability to generalize to unseen configurations and thereby reducing overall errors. . | 36 |

|      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |    |
|------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 2.14 | Comparison between reference temperature field solution and predictions generated by the MT-DeepONet, for four representative geometries parameterized by number and location of the holes. The prediction shows good agreement with the reference across the domain except for some regions around the holes. We used the parametric representation for the target geometry by providing $\{n, p\}$ as an input to the branch network for this study and augmented the low-fidelity input data by applying the masking operation on the solution. . . . . | 37 |
| 3.1  | Schematic of the domain used for solving compressible Euler equations. The blue lines are defined as supersonic inflow boundary condition. The red lines are defined as extrapolation boundary. The black line is a semi-ellipse where the wall-slip boundary conditions are defined. . . . .                                                                                                                                                                                                                                                              | 47 |
| 3.2  | Geometry of all the semi-ellipses used for training and testing. We have used 28 for training and 8 for testing. The blue lines refer to training samples and the dashed red lines refer to testing samples. . . . .                                                                                                                                                                                                                                                                                                                                       | 48 |
| 3.3  | Initial mesh configuration and adaptive mesh at final time. (a) Mesh case 1 at $t = 0$ , (b) Mesh case 3 at $t = 0$ , (c) Mesh case 5 at $t = 0$ , (d) Mesh case 1 at $t = 1$ , (e) Mesh case 3 at $t = 1$ , (f) Mesh case 5 at $t = 1$ . . . . .                                                                                                                                                                                                                                                                                                          | 49 |
| 3.4  | Solution fields for case 1 at $t = 1$ . (a) Density contours, (b) Pressure contours, (c) x-Velocity ( $u$ ), (d) y-Velocity ( $v$ ). . . . .                                                                                                                                                                                                                                                                                                                                                                                                               | 50 |

|     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |    |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 3.5 | (a) Schematic of the domain used for solving compressible Navier-Stokes equations. A capsule geometry is placed inside a rectangle. Supersonic inflow boundary conditions are imposed at the blue boundaries. Supersonic outflow is also imposed at the red boundary. Isothermal wall boundary condition is imposed on the capsule surfaces. A zoomed-in view of the capsule is shown inside the dashed outline box. The geometry of the capsule is parameterized using $\alpha \in [40, 60]$ and $R \in [13.06, 20]$ and the flow condition is parametrized using Mach number as $M \in [5, 8]$ . All the dimensions are in meters and $H = 0.03177$ . (b) Domain discretization using quadrilateral elements . . . . .                                          | 51 |
| 3.6 | Geometry of all capsules used for operator network learning. We use 48 samples for training and 12 for testing, see appendix A2.1. The blue lines refer to training samples and the dashed red lines refer to testing samples.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | 52 |
| 3.7 | (a) Schematic of the domain used for solving compressible Navier-Stokes equations. The domain consists of a converging-diverging nozzle and a duct. The length of the nozzle is 10 and the length of the duct is 10. The throat diameter is set at 2. The inflow Mach number is 0.6 and the Reynolds number is 1000 based on a reference length scale of one. The wall boundary condition is isothermal and $T_w = 2T_\infty$ . The walls of the nozzle are represented using a fifth-order polynomial by imposing sixth geometric constraints. For this problem, we use nozzle inlet diameter $h_i$ , nozzle outlet diameter $h_o$ and the location of the throat $x_t$ as the geometric parameters. (b) A sample discretization of the physical domain. . . . . | 54 |
| 3.8 | Geometry of all the converging diverging nozzles used for operator network learning. We use 42 samples for training and 18 for testing, see appendix A2.1 for dataset details. The blue lines refer to training samples and the dashed red lines refer to testing samples. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | 55 |

|      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |    |
|------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 3.9  | Fusion-DeepONet schematic. All the tensor dimensions are shown. $BH/I$ refers to the branch $I^{\text{th}}$ hidden layer and $TH/I$ denotes trunk net $I^{\text{th}}$ hidden layer. $256^2$ denotes the number of grid points used in the uniform grid. For the irregular grid, the number of points is the maximum point count in the training dataset. $N$ denotes number of training samples. . . . .                                                                                                                                                                                                                                        | 58 |
| 3.10 | Prediction of Fusion-DeepONet for three unseen samples. Each row depicts $u$ and $v$ prediction, ground truth, and point-wise error for an unseen sample on uniform grid. The first two columns show the predicted $u$ and the ground truth. The last two columns depict the predicted $v$ and ground truth. Row 1: test case 1, Row 2: test case 5, Row 3: test case 8. See appendix A2.1 for the values of input parameters for cases 1,5 and 8. . .                                                                                                                                                                                          | 66 |
| 3.11 | Comparison of density prediction using Fusion-DeepONet, U-Net, and FNO neural operators for the unseen case 2 on uniform Cartesian grid. Each row corresponds to a specific neural operator. First row: Fusion-DeepONet, Second row: U-Net, and Third row: FNO. First column shows prediction by the neural operator and second column shows the ground truth with a similar color range as the prediction case. Third column shows absolute value of point-wise error. The color bar for the point-wise error is the same for various neural operators for comparison. See appendix A2.1 for value of input parameters of test case 2. . . . . | 67 |
| 3.12 | Loss versus epochs for Fusion and Vanilla-DeepONets uniform grid. . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | 68 |

|      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |    |
|------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 3.13 | Comparison of $u$ prediction using MeshGraphNet, Vanilla-DeepONet, and Fusion-DeepONet neural operators for the unseen case 5 on an irregular unstructured grid. Each row corresponds to a specific neural operator. First row: MeshGraphNet, Second row: Vanilla-DeepONet, and Third row: Fusion-DeepONet. First column shows prediction by the neural operator and second column shows the ground truth. Third column shows absolute value of point-wise error. The color bar for the point-wise error is the same for various neural operators for comparison. See appendix A2.1 for value of input parameters of test case 5. . . . . | 69 |
| 3.14 | Prediction of Fusion-DeepONet for three unseen samples on irregular unstructured grids. Each row depicts $\rho$ and $p$ prediction and ground truth for an unseen sample. the first two columns show the predicted $\rho$ and the ground truth. The last two columns depict the predicted $p$ and ground truth. Row 1: case 1, Row 2: case 5, Row 3: case 8. See appendix A2.1 for the values of input parameters for test cases 1,5 and 8 . . . . .                                                                                                                                                                                      | 70 |
| 3.15 | Loss versus epochs for Fusion and Vanilla-DeepONets on irregular grid. .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | 70 |
| 3.16 | Energy spectrum for modes of trunk network hidden layers outputs trained with uniform grid training dataset for Vanilla and Fusion-DeepONet. . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | 71 |
| 3.17 | Energy Spectrum of trunk network's hidden layers outputs of Fusion-DeepONet (FD) and Vanilla-DeepONet (VD) for irregular unstructured grid. Each curve represents the Eigen spectrum of extracted data on an individual grid in the training dataset. . . . .                                                                                                                                                                                                                                                                                                                                                                             | 73 |

|      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |    |
|------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 3.18 | Box plot showing sample-wise error comparison for all architectures and grid types. Vanilla-DeepONet and MeshGraphNet frameworks show highest sample-wise error distribution across both mesh types. Error values for u-velocity prediction show the lowest error across all the four outputs predicted in this study. U-Net and Fusion-DeepONet show better error values across all four output predictions. Only the Fusion-DeepONet provides a good error estimate across both regular and irregular grid setup. . . . .               | 74 |
| 3.19 | Two unseen test cases of flow field around hypersonic reentry vehicle predicted by Fusion-DeepONet. (a) Temperature case 4, (b) Temperature case 11. The Fusion-DeepONet was trained using 48 samples and tested for 12 unseen samples. The loss function is MSE of temperature field. from the left first contour plot is prediction, second plot is ground truth, and third plot is the point-wise error defined as $T_{err} =  T_{pred} - T $ , See appendix. A2.1 for the values of input parameters for test cases 4 and 11. . . . . | 76 |
| 3.20 | (a) $L^2$ norm of temperature predictions for testing dataset during the training of Fusion-DeepONet (b) $L^2$ norm of temperature gradient magnitude computed using least square differentiation operator for testing dataset during the training of Fusion-DeepONet. (DEL) refers to derivative enhanced loss function. . . . .                                                                                                                                                                                                         | 78 |
| 3.21 | Heat flux prediction using various loss functions for test samples. GT: Ground Truth, DD: Discrete Derivative operator, LSD: Least Square Differentiation, AD: Automatic Differentiation, NO DEL: No derivative term in the loss function, P3: refers to coarse mesh with polynomial order 3, P7: refers to fine mesh with polynomial order 7. . . . .                                                                                                                                                                                    | 80 |

- 3.22 Two unseen test cases of converging-diverging nozzle flow field predicted by Fusion-DeepONet. All the variables are non-dimensional except temperature which is reported in Kelvin unit ( $K$ ). (a) Density case 1, (b) Density case 12, (c) Pressure case 1, (d) Pressure case 12, (e) Temperature case 1, (f) Temperature case 12, (g)  $v$  case 1, (h)  $v$  case 12, (i)  $u$  case 1, (j)  $u$  case 12. The Fusion-DeepONet was trained using 42 samples and tested for 18 unseen samples. In each sub-figure, from the top, the first plot shows prediction, second shows ground truth, and third shows point-wise error defined as  $E = |\phi_{pred} - \phi|$  where  $\phi$  denotes the primitive variables. See appendix. A2.1 for the values of input parameters for test cases 1 and 12. 81
- 4.1 Representative samples of bottle geometries generated for this study. Top row represents two-parameter bottle samples while the bottom row shows four-parameter bottle samples. Four-parameter bottle family includes higher geometric variability due to a larger design space  $\mathcal{D}$  and poses a more challenging setup for training a neural surrogate. . . . . 93
- 4.2 Visualization of design space  $\mathcal{D}$  for the two bottle families. Here, circles denote the training samples while squares denote the test samples chosen in this study. (a) The two parameter design uses a uniform sampling strategy to cover the design space  $\mathcal{D} \subset \mathbb{R}^2$ . The missing samples in the design matrix are a result of bottle geometries that failed to converge in simulation. (b) The four-parameter design uses Adaptive sampling to efficiently cover the design space  $\mathcal{D} \subset \mathbb{R}^4$  while minimizing computational cost. Note that the four-parameter design space results in more pockets of under-sampled regions. 94

|     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |     |
|-----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 4.3 | DeepONet-Transolver framework for simultaneous prediction of nodal displacements $\mathbf{u}(\mathbf{x}) \in \mathbb{R}^{N \times 3}$ and time-dependent reaction forces $F_R(t)$ for the top load compression analysis of packaging bottles. Here, $N$ represents the number of nodal points that defines a point cloud for geometry $\Omega_i$ , and $C$ represents the input feature dimension, which comprises of the coordinate locations and surface normals. $D$ represents the width of the MLP layer, $H$ represents the number of attention heads for self-attention, $D_h$ represents the attention head dimension, and $S$ represents the number of slices into which the inputs features are distributed based based on their proximity. The Transolver network predicts the nodal displacement $\mathbf{u}(\mathbf{x})$ at the last time step $t = 1$ in the simulation while the integrated DeepONet is used for predicting the reaction force using feature embeddings learnt during Transolver training as input to its Branch network. Both networks are trained concurrently. . . . . | 100 |
| 4.4 | Plot showing distribution of displacement values for a data sample. A log scale is used for the vertical axis to ensure data is scaled appropriately to show the peak of data frequency around displacement values of 0.0 for $u_x$ and $u_y$ . Such unbalanced distribution can bias the average $L^2$ error estimates.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | 105 |
| 4.5 | Point-wise absolute errors in displacement $u_z$ for four-parameter bottle family test samples. Points in red indicate model prediction that is below the minimum displacement value of reference solution while the points in green indicate predictions that are larger than the maximum value of reference solution. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | 106 |
| 4.6 | Scatter plot showing goodness-of-fit of the model for the two bottle design class. The plots show a strong linear correlation between the model's predicted displacements and the target values for all three spatial components of displacement. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | 107 |



|     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |     |
|-----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 4.7 | Comparison of predicted and reference reaction forces versus time for two-parameter (top-row) and four-parameter (bottom-row) bottle designs. The plots show a good agreement between our model's predictions (orange) and the ground truth data (purple). The model accurately captures the mechanical response across all designs, including the complex nonlinear behavior and buckling events evident in the four-parameter designs. . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | 108 |
| 6.1 | Schematic of a vanilla Physics-Informed Neural Network. The unknown solution for a given PDE, denoted by $f(x, t; \Theta)$ , is obtained as an output of the neural network. Automatic differentiation is used to get the necessary derivatives of the output with respect to its inputs $(x, t)$ . These derivatives are then used to calculate the residual loss (denoted as $\mathcal{L}_{res}$ from the original PDE of system and any boundary or initial condition loss (denoted by $\mathcal{L}_{BC/IC}$ ). The total loss ( $\mathcal{L}_{total}$ ) is obtained as a sum of the two loss terms, $\mathcal{L}_{res}$ and $\mathcal{L}_{BC/IC}$ . The term $\lambda$ represents a weighing factor to balance the loss terms during optimization. Typical loss functions used include the mean squared loss, MSE. The network parameters are updated until the convergence criteria, $\epsilon$ is reached. Consequently, the required solution function $f(x, t; \Theta)$ is obtained. See [2] for further details. . . . . | 119 |
| 6.2 | Vanilla DeepONet architecture showing branch and trunk networks (modified from [3]). The branch network takes functions $u^{i(1...n)}$ as inputs evaluated at $m$ sensor locations while the trunk network takes continuous coordinates and parameters, $\mathbf{y}$ as inputs. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | 121 |
| 6.3 | DeepONet surrogate training and testing geometries generated from random sampling from parameter space $\eta_g$ . Note that the numbers correspond to NACA airfoils and duplicate numbers are due to rounding to the nearest integer for readability. The true NACA parameters are drawn from a uniform distribution and are real-valued. Adapted from [4]. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | 126 |

|     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |     |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 6.4 | Use case scenario for 2D airfoil design. The scenario contains expected steps that a user will undertake during a 2D airfoil design process. MyCrunchGPT responds as an assistant during this process to guide the designer while collecting necessary information required for completing user requests. The MyCrunchGPT assistant is instructed to respond to a user entry with an appropriate human-like response. . . . .                                                       | 129 |
| 6.5 | Generic responses generated by ChatGPT when the GPT assistant is not primed with instructions for a specific use case. Here, when a user asks ChatGPT to help with 2D airfoil design, a generic response is received. However, for a specific task, such generic responses may not lead to actionable feedback for the designer. Note that this response is generated by the ChatGPT web-based interface ( <a href="https://chat.openai.com">https://chat.openai.com</a> ). . . . . | 130 |
| 6.6 | Example of instruction used to tailor ChatGPT's response to specific task of designing 2D airfoil based on user query. These instructions are fed to the message history inside ChatGPT thus providing the language model with context for these instructions. This enables us to tune the ChatGPT response to be specific to our tasks. . . . .                                                                                                                                    | 131 |
| 6.7 | Schematic showing workflow for MyCrunchGPT design assistant. Instructions for priming MyCrunchGPT are determined by the sequence of steps outlined in the use-case for airfoil design, as shown on the left side here. These instructions enable receipt of specific response to a user query. The user responses are parsed into required parameters for performing the actions necessary by calling respective python modules. . . . .                                            | 133 |
| 6.8 | <b>[Request 1]</b> MyCrunchGPT's response to user query to design 2D airfoil asking for parameter values $m$ and $p$ as well as total number of new designs that are required. Compare how the response has been tailored for our use-case vs a generic response generated by ChatGPT shown in figure 6.5.                                                                                                                                                                          | 134 |

|      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |     |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 6.9  | <b>[Request 2]</b> Parameter values provided by user which are parsed by MyCrunchGPT to provide feedback to user and initiate new airfoil design process by invoking the necessary python function. MyCrunchGPT provides continuous process feedback to the user about progress. . . . .                                                                                                                                                                                                                 | 134 |
| 6.10 | <b>[Request 3]</b> User requests MyCrunchGPT to show new airfoil designs generated in the design process. Newly generated airfoil designs are displayed in the webapp's image viewer. . . . .                                                                                                                                                                                                                                                                                                            | 135 |
| 6.11 | <b>[Request 4]</b> User requests MyCrunchGPT to analyze flow field around newly designed airfoils. Using this request, MyCrunchGPT calls the necessary python function for analyzing flow fields by calling pre-trained DeepONet. This process takes some time depending on the number of new designs to be evaluated but is significantly less than running a CFD solver. User is provided with continuous feedback while the new designs are being evaluated through the pre-trained DeepONet. . . . . | 136 |
| 6.12 | <b>[Request 5]</b> User requests MyCrunchGPT to show density flow field around airfoil 3. The requested density flow field is plotted and displayed.                                                                                                                                                                                                                                                                                                                                                     | 137 |
| 6.13 | (a) <b>[Request 6]</b> User requests MyCrunchGPT to initiate optimization process to generate the best profile to maximize lift to drag ratio. MyCrunchGPT calls necessary python modules in the back-end and initiates the optimization process, providing regular process updates through an interface showing how the design is evolving with each iteration. (b) Airfoil geometries being updated during the optimization process displayed to user.                                                 | 137 |
| 6.14 | <b>[Request 7]</b> User requests MyCrunchGPT to show the optimization landscape to analyze design space exploration during optimization. MyCrunchGPT invokes the visualization module to display the required plot to user in the image viewer. . . . .                                                                                                                                                                                                                                                  | 138 |

|      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |     |
|------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 6.15 | <b>[Request 8]</b> User request MyCrunchGPT to analyze the optimized airfoil geometry with a CFD tool. MyCrunchGPT then provides the user with the default settings that will be used for the CFD solver, such as number of iteration steps (NumSteps). MyCrunchGPT then waits for the user’s response to determine if the CFD solver settings need to be modified or not.                                                                                                                                                                                                                                                                                                                                   | 138 |
| 6.16 | (a) Updated CFD settings generated by MyCrunchGPT based on changes requested by the user. MyCrunchGPT reads the default CFD settings file provided through GPT instructions, identifies the correct fields (‘NumSteps’ and ‘IO_CheckSteps’ in this case) that need to be changed and then updates it with the corresponding user input. Thereafter, MyCrunchGPT displays the updated settings for CFD simulation to the user for user verification (complete MyCrunchGPT modified simulation settings script not shown here for brevity). (b) Once CFD settings are confirmed by user, MyCrunchGPT invokes the Nektar++ API to initiate simulation with constant feedback about simulation progress. . . . . | 139 |
| 6.17 | <b>[Request 9]</b> User queries MyCrunchGPT to show flow field comparisons. Here, the y-velocity comparison between Nektar++ CFD simulation and DeepONet prediction for the optimized geometry is shown in the image viewer. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | 140 |
| 6.18 | Comparison for x-velocity between CFD simulation (6.18a) and DeepONet prediction (6.18b). When the user requests to see y-velocity comparison, MyCrunchGPT compiles the necessary visualization APIs to generate the comparison: CFD simulation (6.18d), DeepONet predictions (6.18e), and their absolute error (6.18f). These figures here are intended for reader’s clarity and evaluation hence plotted outside the image viewer window. . .                                                                                                                                                                                                                                                              | 141 |

|      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |     |
|------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 6.19 | The user requests MyCrunchGPT to compare the specifications of the optimized airfoil geometry with existing airfoil designs found in literature. MyCrunchGPT searches for relevant airfoil designs with a drag to lift ratio of 0.2 and provides a relevant response that can be investigated further by the user. . . . .                                                                                                                                                                                                                                                                                                                                                     | 142 |
| 6.20 | Discretization of (a) square (b) triangular domain by using unstructured triangular elements used for solving the equation (6.5.1) to validate the velocity fields obtained from PINNs. Equation (6.5.1) is solved by spectral element method provided by Nektar++ frameworks. The boundaries of the domain are marked by respective boundary condition used in formulating the spectral element method. Here $u$ and $v$ represent the horizontal and vertical velocity of the fluid, respectively. . . . .                                                                                                                                                                   | 143 |
| 6.21 | Workflow diagram for asking MyCrunchGPT to solve a PDE with a PINN. The user defines the PDE and relevant details, and asks to train a PINN for the PDE. The user may provide various requests, such as specifying configurations, visualizing different components of the solution, verifying against known solutions/DNS, or saving/loading trained parameters. The user may ask to modify the problem, referring to the previous instructions without explicitly re-defining the PDE. GPT is used to parse the instructions into components, map those components to the corresponding internal modules, and synthesize the full program by composing the selected modules. | 143 |
| 6.22 | Asking MyCrunchGPT to use a PINN to solve the lid-driven cavity CFD benchmark problem. The user specifies the PDE to solve, along with the geometry and boundary conditions in natural language, as well as training configurations. The user also requests to visualize the solution in two different ways, one involving validation against a specific known solution. MyCrunchGPT parses the request into the necessary components and synthesizes and executes a program to complete the request. . . . .                                                                                                                                                                  | 144 |

|      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |     |
|------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 6.23 | MyCrunchGPT interface with the first half of the second requested visualization (visualizing the PINN solution). . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | 144 |
| 6.24 | MyCrunchGPT interface with the second half of the second requested visualization (validation with specified known solution from [5]). Currently, MyCrunchGPT’s capability is limited to extracting contextual meanings from references provided in validation stage. A human-in-loop is required for extracting relevant data from images and tables in the specified reference shown here. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | 145 |
| 6.25 | Asking MyCrunchGPT to solve a variation of the previous problem. The user requests to change the geometry and training iterations, verify with any DNS solution available, and save the learned parameters, for which MyCrunchGPT modifies and executes the code appropriately. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | 145 |
| 7.1  | Schematic for LLM-based multi-agent workflow, consisting of three LLM agents namely <i>Graph Ontologist</i> , <i>Design Engineer</i> , <i>Systems Engineer</i> , each assigned a designated set of tasks in this process. The Design and Systems Engineer agents are provided with their respective knowledge graphs tailored specific to their roles. The Design Engineer agent possesses a set of engineering tools to help him accomplish his tasks. The Systems Engineer agent utilizes a multi-modal language model for reviewing airfoil shapes generated by the Design Engineer and provides design improvement suggestions. The Design Engineer then acts on the feedback, updates the design, and sends it for another round of review process. This process also allows for a Manager (human user in this case) to provide feedback for design improvement. The design-review cycle terminates once the Manager determines the design to be a suitable candidate for further optimization. Details for each component and steps in this workflow are discussed later in Section 7.4. . . . . | 152 |

|     |                                                                                                                                                                                                                                                                                                                                                                                                                                                             |     |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 7.2 | Prompts provided to Graph Ontologist for generating two different KGs for our engineering agents. For the Systems Engineer KG, a more generic set of goals leads to the creation of a larger domain of knowledge as compared to the Design Engineer’s KG, which is more focused on aspects related to airfoil design. . . . .                                                                                                                               | 161 |
| 7.3 | Representation of Knowledge Graphs generated by the Graph Ontologist for the two engineering agents. Systems Engineer’s KG contains $\approx 700$ nodes while the Design Engineer’s KG contains $\approx 50$ nodes. Note that these graphs only contain nodes with a degree of 10 or more, the remaining nodes were filtered out for compactness of the knowledge base. . . . .                                                                             | 162 |
| 7.4 | Initial airfoil samples generated by the Design Engineer agent using the NACA generator and plotter tool. The image shows 10 out of the 100 samples generated in this step following parameter selection in Stage 2. .                                                                                                                                                                                                                                      | 173 |
| 7.5 | Initial filtered design samples selected for design review by the Systems Engineer agent. . . . .                                                                                                                                                                                                                                                                                                                                                           | 173 |
| 7.6 | Final candidate design chosen for the optimization process after the iterative design phase. Note that the camber value for this design exceeds the 6% criteria defined by the Manager at design iteration 1. However, due to the new request from the Manager to improve $C_l$ values further, the agents identify the need to increase the camber values beyond this range to meet the competing requirements of higher lift and pitching moment. . . . . | 180 |

|     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |     |
|-----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 7.7 | Comparison between optimized design and initial starting design selected through the iterative design process. The optimized design has performance values $C_l = 0.811$ , and $C_d = 0.0044$ against the initial starting design with $C_l = 0.8077$ and $C_d = 0.0061$ . Note that since the optimization process uses an initial design that satisfies (to a large extent) both functional and non-functional requirements, the end design does not suffer from qualitative shortcomings such as thin trailing edge which may lead to other challenges in manufacturing and robustness. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | 181 |
| 8.1 | Schematic for LLM-based set-based design workflow with risk filtering strategy. The LLM workflow consists of four agents: <i>Design Engineer</i> , <i>Systems Engineer</i> , <i>Coding Assistant</i> , and <i>Analyst</i> , each assigned a designated role and task. The workflow utilizes different tools that are developed by the Coding Assistant following human manager’s instructions. The workflow starts with the human manager providing a design problem to the Design Engineer. The Design Engineer, equipped with a set of engineering tools, then proceeds to developing, analyzing, and filtering design candidates based on certain rules. After a sequence of design filtering and improvements, an iterative design review and improvement cycle commences with the Systems Engineer reviewing and rating the design solutions based on a utility score, in comparison with a benchmark airfoil design. Final design candidates are reviewed jointly by the Systems Engineer and human manager, who determines which designs to retain in the design process. The process ends once the human manager determines a set of viable candidate designs and requests for CFD analysis of these candidates. . . . . | 187 |
| 8.2 | Sample instruction provided to the Coding Agent by the human user to create a parameter sampling tool based on specific process requirements.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | 194 |



|      |                                                                                                                                                                                                                                                                                                                                                                                          |     |
|------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 8.3  | Plot showing the starting design set and the selected designs (in red) after the utility score-based filtering process. X and Y axes show the Principal Components of the CST parameters that define airfoil geometry. Principal Component values are used here for visualization purpose. . . . .                                                                                       | 207 |
| 8.4  | Sensitivity analysis report generated by the Analyst agent using results from Sobol analysis. This heuristic is used by the Design agent for design modifications $\mathcal{M}_k$ . . . . .                                                                                                                                                                                              | 209 |
| 8.5  | Comparison between cumulative distribution of $C_D$ and $C_L$ before and after design modification by the Design agent. The cumulative distribution shows the improvement in overall values of $C_D$ and $C_L$ after parameter modification based on the Sensitivity analysis. . . . .                                                                                                   | 210 |
| 8.6  | Plot showing $C_L$ distributions of two design candidates: ID-98 (retained) and ID-878 (removed) used for $\alpha$ -risk filtering strategy. The orange line marks the $C_L$ value at $1 - \alpha$ percentile of the distribution. . . . .                                                                                                                                               | 211 |
| 8.7  | Coefficient of Pressure plots for two airfoil samples generated after analysis using the DeepONet surrogate. These pressure plots allows the Systems Engineer agent to perform a comprehensive review of the airfoil design, to include choice of design parameters, integral coefficients such as $C_D, C_L$ , and $C_M$ , and the pressure distribution curves generated here. . . . . | 213 |
| 8.8  | Instruction set provided to the Systems Engineer during the automated design evaluation and filtering phase. . . . .                                                                                                                                                                                                                                                                     | 214 |
| 8.9  | Example of feedback generated by the Systems Engineer during automated design evaluation stage for an invalid design. . . . .                                                                                                                                                                                                                                                            | 215 |
| 8.10 | Example of feedback generated by the Systems Engineer during automated design evaluation stage for a valid design. . . . .                                                                                                                                                                                                                                                               | 216 |
| 8.11 | Feedback for design ID-486 generated by the Systems Engineer with human Manager during first iteration. . . . .                                                                                                                                                                                                                                                                          | 218 |

|      |                                                                                                                                                                                                                                                                                                                                                                                    |     |
|------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 8.12 | Feedback for design ID-486 generated by the Systems Engineer with human Manager during second iteration after design modification. . . . .                                                                                                                                                                                                                                         | 219 |
| 8.13 | Plot showing the design filtering during iterative design and review process. The x and y-axes show the two principal components for the CST parameters for each design. In the iterative phase, the designs are filtered based on the human Manager's input along with feedback received from the Systems Engineer agent. . . . .                                                 | 220 |
| 8.14 | Coefficient of pressure plots predicted by the agentic framework and the corresponding pressure distribution plot from OpenFoam simulation for the final four design candidates. . . . .                                                                                                                                                                                           | 222 |
| A1   | Comparison of MSE for training vanilla DeepONet with various model sizes and L2 regularizer. Vanilla DeepONet is trained for the inviscid hypersonic flow over semi-ellipse problem on uniform grid. The term Vanilla indicates no regularizer and the number 32 or 64 in the legend refers to the number of neurons used in each hidden layer of trunk and branch networks. . . . | 239 |
| A2   | Comparison of regularized POD-DeepONet and not regularized POD-DeepONet MSE values versus epochs. The term "reg" indicates the use of L2 regularizer. . . . .                                                                                                                                                                                                                      | 239 |
| A3   | Parameter conditioned U-Net architecture used for predicting density field across varying geometric shapes. The inputs consist of the geometry parameters (major and minor axes of ellipse) and the binary representation of the geometry. . . . .                                                                                                                                 | 243 |

|     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |     |
|-----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| A4  | Section of the Systems Engineer’s Knowledge Graph showing the two relevant nodes ‘drag coefficient’ and ‘airfoil thickness ratio’ retrieved to extract the required information using the ontological relationship between the two nodes for defining the maximum permissible airfoil thickness. The LLM prompt is used to extract semantically similar chunks from the graph data which is used as additional context for the final LLM response to the task. . . . . | 249 |
| A5  | Section of the Systems Engineer’s Knowledge Graph showing the two relevant nodes ‘Design’ and ‘Manufacturability’ retrieved. The LLM prompt is used to extract semantically similar chunks from the graph data, from which the LLM module also extracts the connection between ‘Manufacturability’ and ‘Thickness’ of the airfoil. . . . .                                                                                                                             | 250 |
| A6  | Loss evolution for the three Bayesian surrogates: $C_D, C_L, C_M$ . . . . .                                                                                                                                                                                                                                                                                                                                                                                            | 260 |
| A7  | Ensemble prediction results from Bayesian surrogate for coefficient $C_D$ . The black error bar shows the expected uncertainty in prediction combining both aleatoric and epistemic uncertainties. . . . .                                                                                                                                                                                                                                                             | 261 |
| A8  | Ensemble prediction results from Bayesian surrogate for coefficient $C_L$ . The black error bar shows the expected uncertainty in prediction combining both aleatoric and epistemic uncertainties. . . . .                                                                                                                                                                                                                                                             | 261 |
| A9  | Ensemble prediction results from Bayesian surrogate for coefficient $C_M$ . The black error bar shows the expected uncertainty in prediction combining both aleatoric and epistemic uncertainties. . . . .                                                                                                                                                                                                                                                             | 262 |
| A10 | Loss evolution for the DeepONet showing convergence. . . . .                                                                                                                                                                                                                                                                                                                                                                                                           | 263 |
| A11 | Samples of DeepONet prediction for coefficient of pressure, $C_P$ . . . . .                                                                                                                                                                                                                                                                                                                                                                                            | 263 |
| A12 | Pairwise plot showing the individual effect of changing CST parameters on aerodynamic coefficients $C_D, C_L$ , and $C_M$ . . . . .                                                                                                                                                                                                                                                                                                                                    | 264 |
| A13 | Plot showing magnitude of Sobol indices showing individual CST parameter effect ( $S_i$ ) and the total effect ( $S_{Ti}$ ) for each aerodynamic coefficient. . .                                                                                                                                                                                                                                                                                                      | 264 |

|     |                                                                                                                                                                                                                                                                                                                                                                                                                                                            |     |
|-----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| A14 | Plot showing the coefficient of pressure plot for benchmark airfoil RAE2822 at $Re=6.3$ million, $Ma=0.6$ , and $AoA=2.57$ degrees [6]. The airfoil profile is shown alongside. The Systems Engineer agent compares the $C_P$ plot along with utility score for the new airfoil designs against this benchmark and determines whether the design should be considered valid or not based on a set of heuristic rules defined by the human Manager. . . . . | 265 |
| A15 | Feedback for design ID-486 generated by the Systems Engineer with human Manager during third iteration after design modification. . . . .                                                                                                                                                                                                                                                                                                                  | 266 |
| A16 | Feedback for design ID-486 generated by the Systems Engineer with human Manager during fourth iteration after design modification. . . . .                                                                                                                                                                                                                                                                                                                 | 267 |
| A17 | Plot showing velocity fields for the final design candidates selected during the agentic workflow. . . . .                                                                                                                                                                                                                                                                                                                                                 | 268 |

# List of Tables

|     |                                                                                                                                                                                                                                                                                                                                                                                                            |    |
|-----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 2.1 | Details of reaction term $F(u)$ from literature and their modified forms used in this study [7]. . . . .                                                                                                                                                                                                                                                                                                   | 18 |
| 2.2 | Relative $\mathcal{L}_2$ error for TL-DeepONet for different source models vs number of samples used during fine-tuning of the target domains, <b>T1</b> and <b>T2</b> . . .                                                                                                                                                                                                                               | 25 |
| 2.3 | Error values for different sample sizes used to fine-tune the target model under the transfer learning scheme for the target domain, <b>T3</b> . The table presents the results for two transfer learning scenarios, each differing in the layers selected for fine-tuning. . . . .                                                                                                                        | 30 |
| 3.1 | Architecture details for all variants of DeepONets . . . . .                                                                                                                                                                                                                                                                                                                                               | 63 |
| 3.2 | Relative $L^2$ norm of the primitive variables errors for different frameworks showing the accuracy achieved. GS refers to Grid Spacing, #P refers to number of trainable parameters. . . . .                                                                                                                                                                                                              | 64 |
| 3.3 | Heat flux prediction accuracy over the unseen dataset using various derivative operators. $\%E$ is average relative error of heat flux predictions over the entire testing dataset. $\%L^2(T)$ is the relative $L^2$ norm of temperature field for all the test cases. The temperature derivative computation during the inference is performed using automatic differentiation for all the cases. . . . . | 79 |
| 3.4 | Comparison of prediction error for the converging-diverging nozzle problem, #P refers to number of trainable parameters. The $L^2$ norm reported is the average of relative $L^2$ norm across 18 unseen samples. . . . .                                                                                                                                                                                   | 82 |

|     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |     |
|-----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 4.1 | Summary of $L^2$ relative errors obtained for the two different bottle families. While the overall errors for test cases are reasonable, few test cases show high errors due to these particular samples being under-represented in the training dataset. Note that the best and worst case error estimates represent error across different data samples. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                        | 104 |
| 6.1 | Name and function of different python APIs used for MyCrunchGPT design assistant framework. Here, each python API executes a pre-defined function from the use-case sequence as defined in figure 6.7. These functions are executed in a sequential order determined by the use-case and input arguments are created by parsing user prompts through MyCrunchGPT interface. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                       | 132 |
| 8.1 | Comparison between aerodynamic performance coefficient prediction using surrogate models at final design iteration against CFD predicted coefficients. We note that the surrogate model provides reasonable estimation of aerodynamic coefficients, $C_D$ and $C_L$ for these four design candidates. However, the epistemic uncertainty ( $\sigma$ ) when evaluating these final designs is high, as compared to the original training dataset. This is a consequence of design modifications performed on the airfoil shapes, which results in the CST parameters of the final candidates residing on the periphery of training data distribution leading to higher epistemic uncertainty. This bias is mitigated, to a certain extent, in the design selection strategy by choosing a composite utility score. . . . . | 222 |
| A1  | Training and test sample selection for 3D plate problem. We select specific radial locations (d) of the holes for training to encompass the innermost and outermost locations as well as the median location. We interpolate the solution on other radial locations during model inference. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | 234 |

|    |                                                                                                                                                                                                         |     |
|----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| A2 | Details of network architecture used for different multi-task problems. . .                                                                                                                             | 235 |
| A3 | Training and testing samples for input parameters of surrogate models of<br>inviscid hypersonic flow over semi-elliptic blunt bodies. . . . .                                                           | 236 |
| A4 | Training and testing samples for input parameters of surrogate models for<br>reentry capsule. . . . .                                                                                                   | 236 |
| A5 | Training and testing samples for input parameters of surrogate models for<br>supersonic flow in the converging diverging nozzle. . . . .                                                                | 237 |
| A6 | Comparison of $L_2$ norms of the error for all the unseen test cases of the<br>LeBlanc problem predicted by the trained 2-step DeepONet [8], UNet [8],<br>Transformer [9], and Fusion-DeepONet. . . . . | 238 |
| A7 | Architecture details for FNO . . . . .                                                                                                                                                                  | 245 |
| A8 | Details of MeshGraphNet architecture. . . . .                                                                                                                                                           | 247 |
| A9 | Airfoil input parameter space showing the range of CST weights $w$ and<br>operating conditions used for our surrogate model training [10]. . . . .                                                      | 260 |

# PART I



# CHAPTER 1

## Introduction to Geometry-Aware Neural Operators

The advent of high-performance computing has established numerical simulation as an important component of scientific discovery. In engineering, disciplines such as computational fluid dynamics (CFD) and finite element analysis (FEA) have become an essential part of the design process, enabling detailed performance analysis and optimization of complex systems prior to building physical prototypes. Despite their benefits, these high-fidelity methods can be resource-intensive, with simulations for certain problems requiring significant computational effort. This bottleneck restricts their application in situations demanding rapid, repetitive evaluations, such as large-scale design space exploration, uncertainty quantification, and the development of real-time digital twins [11].

One way to overcome this computational barrier is to use surrogate modeling, where a data-driven model is trained to approximate the output of the expensive high-fidelity solver. While traditional methods like polynomial chaos expansion [12, 13] and Kriging [14] have been used successfully, the recent growth in deep learning methods has improved our capability in building accurate and sophisticated surrogates, capable of complex mappings. Recently, neural operators have evolved as a subset of deep learning models, which can

learn mappings between infinite-dimensional function spaces [15, 16]. Unlike conventional neural networks that operate on finite-dimensional vectors, a neural operator, denoted as  $\mathcal{G} : \mathcal{X} \rightarrow \mathcal{Y}$ , learns a map from an input function space  $\mathcal{X}$  (e.g., representing initial conditions, boundary conditions, or physical parameters) to an output function space  $\mathcal{Y}$  (representing the solution field) [17]. This grants these operator networks some important properties, including mesh-invariance and the ability to generalize across varying input functions. Prominent architectures like the Deep Operator Network (DeepONet) and the Fourier Neural Operator (FNO) have demonstrated remarkable success as efficient surrogate models in scientific machine learning tasks [18, 19, 20].

However, a fundamental limitation of early neural operator frameworks was their implicit assumption of a fixed geometric domain. Most pioneering work focused on problems where the geometry of the computational domain remained constant, with variations confined to PDE coefficients or boundary conditions. This is a critical shortcoming for the majority of engineering design problems, where the shape of an object, be it an airfoil, a heat sink, or a structural component, is the subject of interest from the design perspective. Training a separate neural operator for every geometric permutation is computationally prohibitive and defeats the purpose of creating an efficient surrogate. The research challenge, therefore, lies in developing a new class of operators that are explicitly geometry-aware, capable of accepting geometric parameters as input and predicting the corresponding physical response across a design space.

The first part of the thesis is dedicated to addressing this geometry challenge, presenting a cohesive body of work on the development and application of geometry-aware neural operators. We introduce examples of geometry-aware operator models and training methodologies designed to provide these models with the ability to generalize across diverse and complex geometric domains, thereby making them useful tools for engineering design. First, we explore the concept of synergistic learning through a multi-task framework. Recognizing that training individual models for each geometry is inefficient, we propose the Multi-Task DeepONet (MT-DeepONet) [21]. This framework is engineered to learn

solutions across multiple distinct geometries and varying PDE parameters within a single, concurrent training session. By introducing architectural modifications that encode geometric information, such as a binary mask representation, MT-DeepONet leverages shared information between related tasks to improve both training efficiency and generalization performance. This approach is demonstrated on a range of problems, including 2D Darcy flow and 3D heat transfer, establishing a foundational strategy for handling parameterized geometries in a unified and cost-effective manner.

Building upon this foundation, we then tackle the dual challenges of data scarcity and complex physics, which are particularly important in fields like aerospace engineering. We introduce Fusion-DeepONet [22], a highly data-efficient and robust operator network specifically tailored for geometry-dependent hypersonic and supersonic flow problems. This architecture is designed to perform accurately on both structured and unstructured grids, a crucial feature for real-world engineering meshes [23, 24]. Furthermore, we enhance its predictive power for critical engineering quantities, such as surface heat flux on a reentry vehicle, by incorporating a derivative-enhanced loss term. Through rigorous evaluation against state-of-the-art models on challenging aerodynamic problems, Fusion-DeepONet demonstrates a superior balance of accuracy, efficiency, and robustness, especially in the low-data regimes typical of high-fidelity simulation campaigns.

Finally, we extend the applicability of our geometry-aware framework to the domain of nonlinear structural mechanics. We present a hybrid DeepONet-Transolver framework [25, 26] to analyze the complex buckling behavior of PET bottles under compressive loads – a problem characterized by large deformations and intricate, non-parametric geometric variations. This hybrid model is uniquely capable of performing multi-task predictions, simultaneously learning the full-field nodal displacements and the time-dependent reaction forces. This work showcases the versatility of our operator-learning approach and also highlights its potential to serve as a rapid design evaluation tool in computational mechanics, accurately capturing pertinent physical phenomena across a diverse set of industrial designs.

In summary, this part of the thesis contributes a suite of neural operator frameworks that systematically overcome the limitation of geometric variance in scientific machine learning. By developing models that are efficient, data-aware, and applicable to the complex physics of both fluids and solids, we lay the groundwork for the next generation of surrogate models poised to revolutionize the computational engineering design process. The subsequent chapters will detail the architecture, implementation, and validation of the MT-DeepONet, Fusion-DeepONet, and the hybrid DeepONet-Transolver frameworks, respectively.

# CHAPTER 2

## Synergistic Learning with Multi-Task DeepONet for Efficient PDE Problem Solving

### 2.1 Introduction

In scientific machine learning, we can solve partial differential equations (PDEs) by finding the solution operator, known as the neural operator (NO). The NO takes different functions as inputs, such as initial and boundary conditions, and maps them to the solution of the PDE. Traditional numerical methods such as finite difference, finite element, and spectral methods are generally used to compute solutions to PDEs. There is an increasing interest in using scientific machine learning methods to solve PDEs in real time across diverse applications. However, these real-time methods can be computationally expensive when dealing with high dimensional PDEs, and incorporating experimental measurement data as model inputs is often not possible. Additionally, the solution must be recomputed for minor changes in the input function or the geometry domain that add to the computational burden for the users.

Recently, Deep neural networks (DNNs) have been employed in NOs [27, 28] to

approximate mappings between infinite-dimensional Banach spaces, in contrast to the finite-dimensional vector space mapping learned through functional regression in conventional DNNs. Frameworks such as deep operator network (DeepONet) [29] and integral operators, which include architectures like the Fourier neural operator (FNO) [30], the wavelet neural operator (WNO) [31], the Laplace neural operator (LNO) [32], and convolutional neural operator (CNO) [33], have demonstrated significant potential over a range of applications. While the early success of NOs has been promising, their predictive performance is often limited by the availability of labeled data for training. Collecting large labeled datasets for each task can be computationally intractable, especially for high-fidelity or multi-scale models. Multi-task learning (MTL) is an alternative mechanism aimed at leveraging useful information from related learning tasks to address data sparsity and overfitting issues [34]. This inductive transfer mechanism trains tasks in parallel while using a shared representation, assuming that the tasks are associated with each other and that shared information among them can lead to synergistic learning performance.

MTL has been explored in traditional machine learning tasks such as natural language processing, computer vision, and healthcare to improve generalization scenarios with limited training data [35, 36, 37, 38]. There are two prevalent techniques for using MTL based on the connections between the learning tasks: hard parameter sharing and soft parameter sharing. Hard parameter sharing uses a common hidden layer for all tasks, while soft parameter sharing regularizes the distance between parameters in different models. Hard parameter sharing techniques are useful when tasks have different input data distributions but similar output conditional distributions (*i.e.*,  $P(\mathbf{x}_s) \neq P(\mathbf{x}_t)$  and  $P(\mathbf{y}_s|\mathbf{x}_s) = P(\mathbf{y}_t|\mathbf{x}_t)$ ), typically referred to as covariate shift. The idea of MTL is a relatively new paradigm in scientific machine learning but one that may yield significant benefits considering the limited data availability for the majority of scientific and engineering applications.

In this regard, Liu [39] introduced an in-context learning paradigm to learn a common operator mapping from a set of differential equations. Liu et al. utilize a transformer

framework where key-value pairs are used as input queries for predicting the output solution of a differential equation. The key-value pair represents conditions that define the differential equation such as the initial condition in a temporal problem. This framework shows good generalization capabilities due to its in-context learning paradigm. Mishra et al. [40] recently proposed POSEIDON, a foundation model for learning the solution to different PDE problems by using an operator network that is pre-trained on multiple PDEs related to fluid dynamics. While both these methods generate accurate results in conventional PDE tasks, a key aspect related to the handling of varying geometric domains has not been addressed. For instance, in POSEIDON, a research opportunity that remains is whether this framework can be extended to learn solutions across multiple geometries in conjunction with the different initial conditions that were used for different pre-training examples. Existing works that explore the idea of operator learning across different domains include [41, 19, 23]. However, these methods have inherent drawbacks when operating on realistic engineering domains where sample size is limited. For instance, [41] is limited by its use of an ‘average’ nodal location for learning operator mapping across multiple geometries that requires the same number of node points across all geometries. This is typically not feasible for geometries that contain design parameters that vary significantly.

In science and engineering, the application of MTL frameworks is challenging, since PDEs with the same initial or boundary conditions can represent vastly different physical systems. However, a group of tasks can share the same marginal distribution of inputs or even the same input function, while their conditional output distributions may differ significantly. This scenario, known as conditional shift, occurs when  $P(\mathbf{x}_s) = P(\mathbf{x}_t)$  and  $P(\mathbf{y}_s|\mathbf{x}_s) \neq P(\mathbf{y}_t|\mathbf{x}_t)$ . In such cases, transfer learning, often referred to as soft parameter sharing, has shown success. In our recent study [1], we proposed the idea of transfer learning within the DeepONet (TL-DeepONet) architecture to enable the knowledge transfer from one task to a related but different task, allowing task-specific learning under conditional shift. However, we found limitations in this framework when attempting to

transfer knowledge across varying geometries, such as changes in internal and external boundaries of the target geometry. In this work, we aim to extend DeepONet’s capability to train multiple parameterized PDEs on multiple domains concurrently, enhancing the generalizability of a single network and thereby improving the transfer learning process to new geometric domains. To achieve this, we introduce multi-task DeepONet (MT-DeepONet) designed to predict solutions for different but correlated tasks in a single training session. Figure 2.1 illustrates the problem statements for the MTL problems considered in this study of MT-DeepONet. The main contributions of this work are summarized as follows:

- **Extension of DeepONet for concurrent training over multiple tasks:** We develop MT-DeepONet to approximate the solution for multiple tasks (different parametric conditions and source terms) simultaneously, without requiring re-training.
- **Improved generalizability:** We investigate the generalization ability of MT-DeepONet for knowledge sharing across different geometries as an extension to [1]. We demonstrate improvement in target model learning across varied geometries using the MTL source model as compared to a single-task source model.
- **Enhanced knowledge transfer:** We introduce a masking operation that enables our MT-DeepONet to learn solutions across varied geometries. Our methodology is demonstrated by learning solutions for 2D Darcy flow equations across multiple geometric domains and steady-state heat transfer in multiple 3D plate designs parameterized by the location and number of heating sources.

The paper is organized as follows. In Section 2.2, we provide a brief review of the original DeepONet framework followed by a description of the proposed MT-DeepONet framework. In Section 2.3, we present a comprehensive collection of problems for which the proposed MT-DeepONet has been extensively studied. We discuss the data generation process and include results and comparisons for multiple examples. Finally, we summarize our observations and provide concluding remarks in Section 8.7 along with some limitations of the framework.



| Application      | Model                                                                                                                                                                                                                          | Varying Conditions                                                                                                                                                                                                                                                                                                          |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Fisher Equations | $u_t = u_{xx} + F(u, a, b)$ $F(u, a, b) = \begin{cases} au(1-u)e^{-b(1-u)} \\ au(1-u) \\ au(1-u^2) \end{cases}$ $\mathcal{G}_\theta: [F(u, a, b), u(x, t = 0)] \rightarrow u(x, t)$                                            | $a \in [0.1, 1.0], b \in [0.5, 2.0]$<br>$u(x, t = 0) = \mathcal{GP}(\mu, \sigma^2)$<br>$\mu = 5 + 0.1 * \sin(\pi * t)$<br>$\sigma^2 = 4, l_x = 0.4$                                                                                                                                                                         |
| Darcy Flow       | $\nabla \cdot (K(\mathbf{x}) \nabla h(\mathbf{x})) = 1, \Omega \subseteq \mathbb{R}^2$ $h(\mathbf{x}) = 0 \quad \forall \mathbf{x} \in \partial\Omega$ $\mathcal{G}_\theta: [\Omega, K(\mathbf{x})] \rightarrow h(\mathbf{x})$ | $K(\mathbf{x}) \sim \mathcal{GP}(0, \mathcal{K}(\mathbf{x}, \mathbf{x}')), \mathbf{x}, \mathbf{x}' \in [0, 1]^2$<br>$\mathcal{K}(\mathbf{x}, \mathbf{x}') = \exp \left[ -\frac{(\mathbf{x} - \mathbf{x}')^2}{2l^2} \right], l = 0.05$<br> |
| Heat Transfer    | $-\nabla \cdot (K \nabla u(\mathbf{x})) = q, \Omega \subseteq \mathbb{R}^3$ $\mathcal{G}_\theta: \Omega \rightarrow u(\mathbf{x})$                                                                                             |                                                                                                                                                                                                                                           |

**Figure 2.1:** A schematic representation of the operator learning benchmarks and MTL scenarios considered in this study.

## 2.2 Multi-task learning in neural operators

Neural operators learn nonlinear mappings between functional spaces on bounded domains, offering a unique framework for real-time solution inference for complex parametric PDEs. Here, ‘parametric PDEs’ refer to PDE systems with parameters that vary over a certain range. Typically, DeepONet (our choice of operator network for this work) is trained on a fixed domain,  $\Omega$ , for varying parametric conditions drawn from a distribution. In this work, we introduce MT-DeepONet, which enables concurrent training of multiple functions (leading to different dynamics) and multiple geometries, along with varying parametric conditions. This section provides a brief overview of the DeepONet architecture and extends it to discuss our multi-task DeepONet framework.

### 2.2.1 Deep operator network

The goal of operator learning is to learn a mapping between two infinite-dimensional spaces on a bounded open set  $\Omega \subset \mathbb{R}^D$ , given a finite number of input-output pairs. Let

$\mathcal{U}$  and  $\mathcal{S}$  be Banach spaces of vector-valued functions defined as:

$$\mathcal{U} = \{\Omega; u : \mathcal{X} \rightarrow \mathbb{R}^{d_u}\}, \quad \mathcal{X} \subseteq \mathbb{R}^{d_x} \quad (2.2.1)$$

$$\mathcal{S} = \{\Omega; s : \mathcal{Y} \rightarrow \mathbb{R}^{d_s}\}, \quad \mathcal{Y} \subseteq \mathbb{R}^{d_y}, \quad (2.2.2)$$

where  $\mathcal{U}$  and  $\mathcal{S}$  denote the set of input functions and the corresponding output functions, respectively. The operator learning task is defined as  $\mathcal{G} : \mathcal{U} \rightarrow \mathcal{S}$ . The objective is to approximate the nonlinear operator,  $\mathcal{G}$ , via the following parametric mapping:

$$\mathcal{G} : \mathcal{U} \times \Theta \rightarrow \mathcal{S} \quad \text{or} \quad \mathcal{G}_\theta : \mathcal{U} \rightarrow \mathcal{S}, \quad \theta \in \Theta, \quad (2.2.3)$$

where  $\Theta$  is a finite-dimensional parameter space. In the standard setting, the optimal parameters  $\theta^*$  are learned by training the neural operator with a set of labeled observations  $\mathcal{D} = \{(u^{(i)}, s^{(i)})\}_{i=1}^N$ , which contains  $N$  pairs of input and output functions. When a physical system is described by PDEs, it involves multiple functions, such as the PDE solution, the forcing term, the initial condition, and the boundary conditions. We are typically interested in predicting one of these functions, which is the output of the solution operator (defined on the space  $\mathcal{S}$ ), based on the varied forms of the other functions, i.e., the input functions in the space  $\mathcal{U}$ .

The deep operator network (DeepONet) is inspired by the universal approximation theorem for operators [42]. The architecture of DeepONet comprises two deep neural networks: the branch network and the trunk network. The branch network encodes the input functions  $\mathcal{U}$  at fixed sensor points  $\{x_1, x_2, \dots, x_m\}$ , while the trunk network encodes the information related to the spatio-temporal coordinates  $\zeta = \{x_i, y_i, t_i\}$  where the solution operator is evaluated. The trunk network takes these spatial and temporal coordinates to compute the loss function. The solution operator for an input realization

$u_1$  can be expressed as:

$$\mathcal{G}_{\boldsymbol{\theta}}(u_1)(\zeta) = \sum_{i=1}^p b_i \cdot tr_i = \sum_{i=1}^p b_i(u_1(x_1), u_1(x_2), \dots, u_1(x_m)) \cdot tr_i(\zeta), \quad (2.2.4)$$

where  $\{b_1, b_2, \dots, b_p\}$  are the output embeddings of the branch network and  $\{tr_1, tr_2, \dots, tr_p\}$  are the output embeddings of the trunk network. In Eq. (2.2.4),  $\boldsymbol{\theta} = (\mathbf{W}, \mathbf{b})$  represents the trainable parameters of the network including weights,  $\mathbf{W}$ , and biases,  $\mathbf{b}$ . The optimized parameters  $\boldsymbol{\theta}^*$ , are obtained by minimizing a standard loss function ( $\mathcal{L}_1$  or  $\mathcal{L}_2$ ) using a standard optimization algorithm.

The DeepONet model provides a flexible framework that allows the branch and trunk networks to be configured with different architectures. For equispaced discretization of the input function, a convolutional neural network (CNN) can be utilized for the branch network architecture, whereas a multilayer perceptron (MLP) is often employed for a sparse representation of the input function. An MLP is commonly used for the trunk network to handle the low-dimensional evaluation points,  $\zeta$ . Since its inception, standard DeepONet has been applied to address complex, high-dimensional systems [43, 44, 3, 45, 46, 47, 48]. Recent extensions for DeepONet have explored multi-fidelity learning [49, 50, 51], integration of multiple-input continuous operators [52, 53], hybrid transferable numerical solvers [54, 55], resolution independent learning [56], transfer learning [57], physics-informed learning to satisfy the underlying PDE [58, 28, 59], and learning in latent spaces [60].

## 2.2.2 Multi-task deep operator network (MT-DeepONet)

Neural operators are inherently data-driven models that require substantial datasets to develop a generalized solution operator for parameterized PDEs. In general, applications using DeepONet to learn the solution operator have focused on single-domain geometries, parameterizing either the source term or the initial condition with a Gaussian random field. The different applications explored in this study are illustrated in Figure 2.1. The

MT-DeepONet framework is designed to:

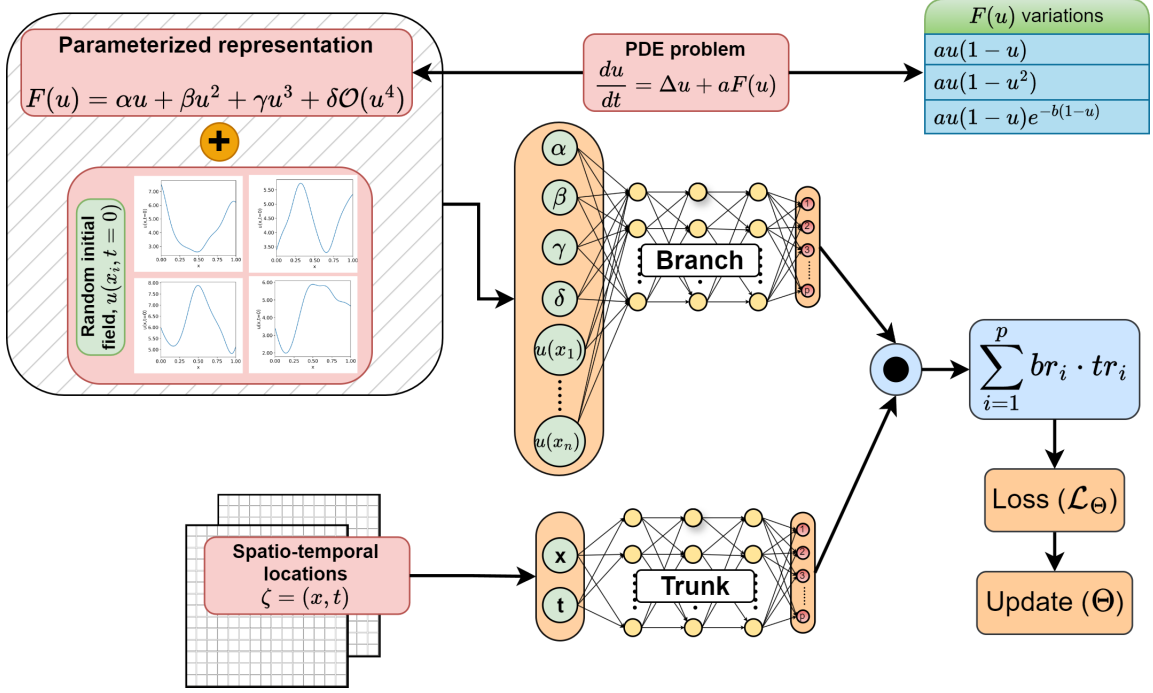
- Learn operator mapping between randomly generated initial conditions and multiple PDE tasks constituted by different forms of the Fisher equation.
- Learn the solution operator on different geometries and initial conditions, thereby improving the source model’s ability to transfer knowledge to a target model. This is illustrated by solving the Darcy flow problem in multiple 2D geometries.

The primary modification in the MT-DeepONet framework occurs in the branch network of the standard DeepONet architecture. Each problem is addressed uniquely, while collectively addressing multi-task learning. For example, in the case of multiple source terms in the Fisher equation (see Figure 2.1), the source terms are represented as a polynomial to create a unique representation for each equation:

$$F(u) = \alpha u + \beta u^2 + \gamma u^3 + \delta \mathcal{O}(u^4). \quad (2.2.5)$$

The coefficients of the dependent variable  $u$  in this polynomial expression are used as inputs to the branch network, along with the random initial conditions that define the problem. A schematic of the framework is shown in Figure 2.2 for understanding. Further details of the problem are discussed in Section 2.3.1.

To accommodate varying geometries concurrently in a single training process, we use a binary mask (array of 0’s and 1’s). This mask is constructed by fitting the geometry within a unit square plate for 2D problems and a box for 3D problems. The masking function assigns a value of 1 to points within the boundary of the desired geometry and 0 to points outside the boundary but within the plate or box. Figure 2.3 illustrates the masking function with a triangular geometry within a square plate. The solution operator is defined as the product of  $\mathcal{G}_\theta$  (as described in Equation 2.2.4) and the binary masking function, ensuring the solution is confined within the geometry’s bounds. Additionally, the binary masking technique ensures the uniqueness of geometries when training a DeepONet model. The binary mask encodes the geometric information to create a bijective mapping between

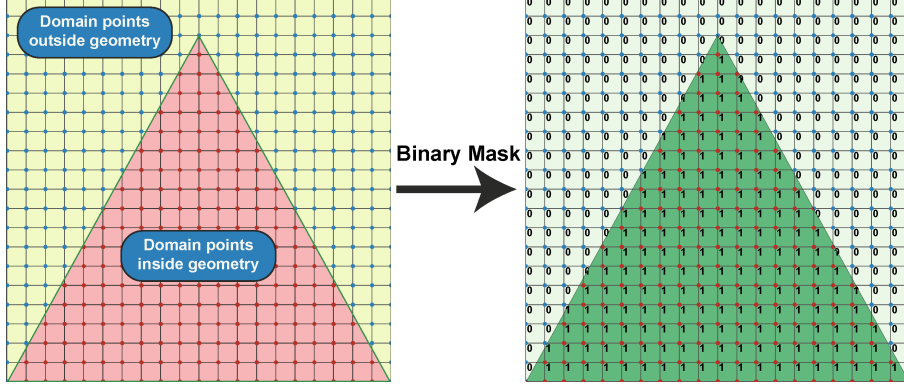


**Figure 2.2:** Schematic of the MT-DeepONet designed to learn a family of parametric PDEs (Fisher equations) defined by different forcing functions,  $F(u)$ . The parametric representation of this equation family, along with random initial condition fields, is input to the branch network. Spatio-temporal points are input to the trunk network. The multi-task operator network learns to predict the solution field,  $u$  across this parameterized family of equations and random initial solutions concurrently.

the network’s input and output spaces. This helps the trunk network learn the underlying solution patterns more effectively, without being confused by the varying geometrical shapes. Additionally, the binary mask provides an alternative to data augmentation when training data is limited and the aim is to generalize the solution across geometries. Without the mask, a much larger dataset would be needed to learn across multiple geometries, which is often impractical. The binary mask makes the DeepONet framework more robust and applicable in real-world scenarios with diverse geometries and limited data availability.

Algorithm 1 details the steps for training the MT-DeepONet with a binary mask to address problems across multiple geometries. This approach also tests our hypothesis that learning multiple geometries improves the source model’s ability to transfer knowledge to target models with different geometries (Section 2.3.2). However, given the complexity of such knowledge transfer in the diverse range of PDEs representing different physical

systems, we demonstrate the effectiveness of the masking framework for learning the solution operator across unseen 3D geometries in a steady-state heat transfer problem (Section 2.3.3).



**Figure 2.3:** Schematic showing the binary masking function for a triangular geometry. A uniform grid  $100 \times 100$  is sampled in  $\Omega \in [0, 1] \times [0, 1]$  and used for generating the basis function in the trunk network. The binary mask is constructed by delineating the boundaries of the domain. Grid points within the domain boundary are denoted by 1, while those outside are denoted by 0. The binary mask is applied to the solution from the operator network, enforcing the solution outside the geometry to be 0, thereby aiding with network convergence. Additionally, the binary mask also establishes sample uniqueness by enforcing the geometric constraint on the output space of MT-DeepONet thereby ensuring a bijective mapping between input and output functional spaces.

---

**Algorithm 1:** Operator learning with binary mask

---

```

1 Prepare the binary mask:  $M_{binary}$ 
2 Input:  $K(x), M_{binary}$ 
3 Output:  $\mathcal{G}(x, K, M)$ 
4 Branch and Trunk network parameters:  $\Theta_{br}, \Theta_{tr}$ 
5 Number of Epochs: n
6 for  $n \leq n_{max}$  do
7    $br_k \leftarrow \mathcal{F}(K(x), M, \Theta_{br})$ 
8    $tr_k \leftarrow \mathcal{T}(x, \Theta_{tr})$ 
9    $\mathcal{G}(x, K, M, \Theta_{br}, \Theta_{tr}) \leftarrow \sum_{k=1}^p br_k \cdot tr_k$ 
10   $\mathcal{G}(x, K, M, \Theta_{br}, \Theta_{tr}) \leftarrow \mathcal{G}(x, K, M, \Theta_{br}, \Theta_{tr}) \cdot M_{binary}$ 
11   $\Theta_{br}, \Theta_{tr} \leftarrow \text{backprop update}$ 
12 end

```

---

MTL serves as an inductive transfer mechanism designed to enhance generalization performance compared to single-task learning. It achieves this by leveraging valuable information from multiple learning tasks and utilizing domain-specific insights embedded within training samples across related tasks. In our study, we demonstrate the effectiveness

of MTL in learning solutions across diverse sets of PDEs, initial conditions, and geometries simultaneously. In [52], the authors propose a multi input-output DeepONet framework to learn the operator mapping tasks across multiple varying fields simultaneously for a single PDE. The architecture, MIONet, considers each varying input field as an individual branch network and is primarily designed to learn a single PDE with multiple varying input conditions, similar to the MT-DeepONet framework. We note that the MIONet framework can be adopted as an alternative to our proposed approach for the Fisher equation problem by using two separate branch networks, one each for: (i) unified PDE coefficients,  $\{\alpha, \beta, \gamma, \delta\}$ , and (ii) initial conditions. However, this approach may increase the number of trainable parameters in the model to achieve generalization across diverse sample sets as compared to our MT-DeepONet approach using a single branch setup. Empirically, we notice a lower accuracy with MIONet (Relative  $\mathcal{L}_2$  Error  $\approx 5\%$ ) as compared to our MT-DeepONet framework (Relative  $\mathcal{L}_2$  Error  $\approx 2\%$ ), while MIONet uses more trainable parameters due to its two-branch construction. We show an empirical comparison between our proposed MT-DeepONet and MIONet framework in Appendix section A1.1.

While both MIONet and MT-DeepONet address multi-input operator learning, the MT-DeepONet framework fundamentally differs in its objectives and capabilities. MIONet was primarily designed to learn a single PDE with multiple varying fields, whereas our approach demonstrates the simultaneous learning of different PDEs and geometries within a single training session. Although one could potentially adapt MIONet to handle the Fischer equation by using a separate branch network for the unified PDE coefficients,  $\{\alpha, \beta, \gamma, \delta\}$ , such an approach would require a substantially larger number of samples to achieve generalization across these parameters. This increased data requirement stems from the fact that these parameters are independent, uncorrelated scalar values, unlike the Gaussian Random fields utilized in MIONet. Our approach is particularly advantageous when identifying specific PDEs where some parameters in  $\{\alpha, \beta, \gamma, \delta\}$  may be zero, allowing for more efficient representation of different equation forms. To that end, Our objective

in this work is to develop a framework that is capable of handling multiple PDE tasks and input conditions simultaneously. An additional goal is to use this framework to learn operator mapping across multiple PDEs and geometries concurrently with varying input conditions.

## 2.3 Numerical examples

In this section, we explore the capabilities of the proposed MT-DeepONet framework on three problems shown in Figure 2.1. Detailed information on data generation for each problem can be found in Appendix A1.2, while specifics about the network architecture are provided in Appendix A1.4.

### 2.3.1 Fisher Equations

The first example considers the Fisher equation proposed by Ronald Fisher in 1937, which provides a mathematical framework for analyzing population dynamics and chemical wave propagation with diffusion [61]. The original reaction-diffusion equation is defined as:

$$u_t = Du_{xx} + ru(1 - u), \quad (2.3.1)$$

where  $u$  represents population density that varies spatially and temporally,  $D$  and  $r$  are scalar parameters denoting the diffusion coefficient and the intrinsic growth rate, respectively. In dimensionless form, this equation is written as:

$$u_t = u_{xx} + u(1 - u). \quad (2.3.2)$$



Kolmogorov, Petrovsky, and Piskunov introduced a more general form, the Fisher-KPP equation [62]:

$$u_t = D(u_{xx} + u_{yy}) + F(u), \quad (2.3.3)$$

where the population density,  $u$ , varies along two spatial dimensions  $(x, y)$ , and the reaction term  $F(u)$  must satisfy the following criteria:

$$F(0) = F(1) = 1, \quad (2.3.4a)$$

$$F(u) > 0, \quad u \in (0, 1), \quad (2.3.4b)$$

$$F'(0) = \alpha, \quad \alpha > 0, \quad (2.3.4c)$$

$$F'(u) < \alpha, \quad u \in (0, 1). \quad (2.3.4d)$$

Assuming that the density  $u$  is invariant along the  $y$ -axis, the Fisher-KPP equation in dimensionless form is re-written as:

$$u_t = u_{xx} + F(u), \quad 0 \leq u \leq 1. \quad (2.3.5)$$

**Table 2.1:** *Details of reaction term  $F(u)$  from literature and their modified forms used in this study [7].*

| Equation                         | General Form<br>$F(u)$        | Our Form<br>$F(u)$     | Parameter<br>Range                       |
|----------------------------------|-------------------------------|------------------------|------------------------------------------|
| Fisher [61]                      | $au(1 - u)$                   | $au(1 - u)$            | $a \in [0.1, 1.0]$                       |
| Newell–Whitehead–Segel [63]      | $au - bu^q, q \in \mathbb{N}$ | $au(1 - u^2)$          | $a \in [0.1, 1.0]$                       |
| Zeldovich–Frank–Kamenetskii [64] | $au(1 - u)e^{-b(1-u)}$        | $au(1 - u)e^{-b(1-u)}$ | $a \in [0.1, 1.0]$<br>$b \in [0.5, 2.0]$ |

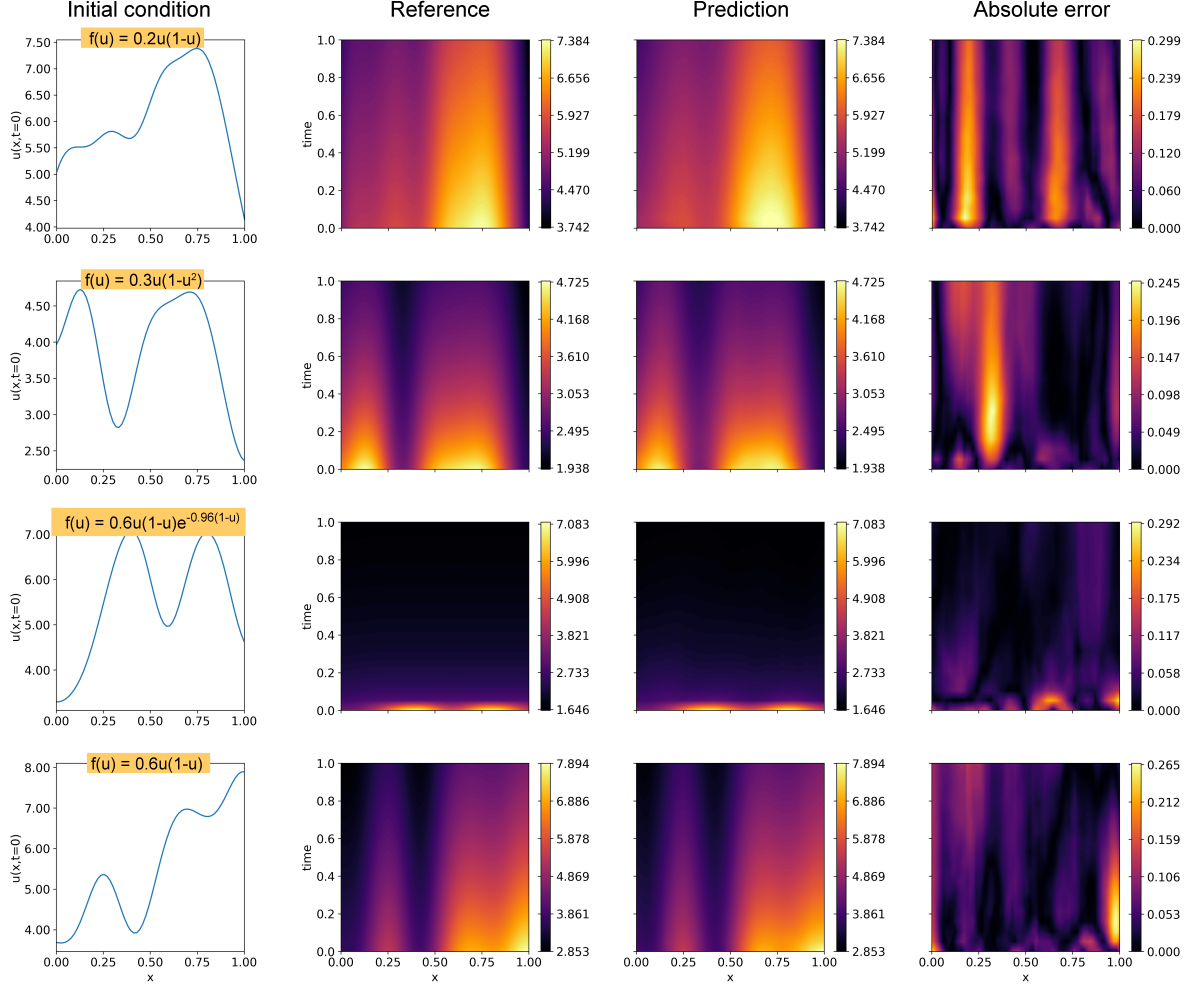
In the conventional operator learning task, the focus is typically on analyzing the change in density profile  $u(x, t)$  over a one-dimensional spatiotemporal domain  $x \in [0, 1]$  and  $t \in [0, 1]$  for parameterized initial condition  $u(x, t = 0)$  drawn from a distribution.

Due to diffusion, regions with higher density expand over time towards areas with lower density, based on the initial distribution. For our MT-DeepONet, we aim to evaluate the Fisher-KPP model to learn the density variation  $u(x, t)$  over time for two tasks: (i) varying initial conditions of density  $u(x, t = 0)$ , and (ii) three reaction functions  $F(u)$  in a single training cycle, considering different functional of  $F(u)$  that are separately parameterized by two scalar coefficients  $a$  and  $b$ . We generate multiple initial conditions as a Gaussian random field and multiple forcing functions  $F(u)$  using the Fisher-KPP general form with three reaction terms from the literature, as listed in Table 2.1.

To incorporate the different functional forms of  $F(u, a, b)$  as inputs to the network, we express the functions as shown in Equation 2.2.5. For the Zeldovich form, we use the Taylor expansion and re-write it as:

$$F(u) = au(1 - u)e^{-b(1-u)} = (au - au^2) \left( 1 - b(1 - u) + \frac{b^2(1 - u)^2}{2} + \mathcal{O}(u^3) \right). \quad (2.3.6)$$

The coefficients of  $u$ ,  $u^2$ , and  $u^3$  are represented as  $\alpha$ ,  $\beta$  and  $\gamma$ , respectively in Figure 2.2, with the constant term is denoted as  $\delta$ . To train the MT-DeepONet for the Fisher-KPP equations, the coefficients of the parameterized form of  $F(u, a, b)$  are concatenated with the flattened initial conditions,  $u(x, t = 0)$  and used as inputs to the branch network. We use Adam optimizer with a progressively reducing learning rate and the network is trained for 100,000 epochs using mean-squared error as the loss function. The relative  $\mathcal{L}_2$  norm for the test samples is approximately  $\sim 2.4\%$ . Figure 2.4 presents four representative test case predictions using the MT-DeepONet. The results indicate that the multi-task operator network can capture large-scale features across the space-time domain with reasonable accuracy, under varying initial conditions and parameters of the Fisher-KPP equation. Additionally, the training time for individual DeepONet for each forcing term with varying initial conditions was 1250 - 1300 seconds on an NVIDIA A100 GPU. In contrast, the MT-DeepONet framework was trained in  $\approx 1200$  seconds, showing the computational effectiveness of our approach.



**Figure 2.4:** Comparison between the reference solution and prediction obtained using the proposed multi-task operator network for representative test cases. The plot shows the operator network predictions against reference solutions for different initial conditions and forcing functions  $F(u)$ . The results demonstrate good overall accuracy across different initial conditions and forcing functions.

### 2.3.2 Darcy Flow in 2D geometries

In the second example, we consider the Darcy flow through a bounded domain and aim to train multiple 2D geometries (bounded domains) simultaneously considering parameterized spatially varying conductivity fields. An earlier attempt, as noted in [65], focused on learning similar yet parameterized geometries in a concurrent training session. In contrast, our work demonstrates the learning of distinctly different geometries.

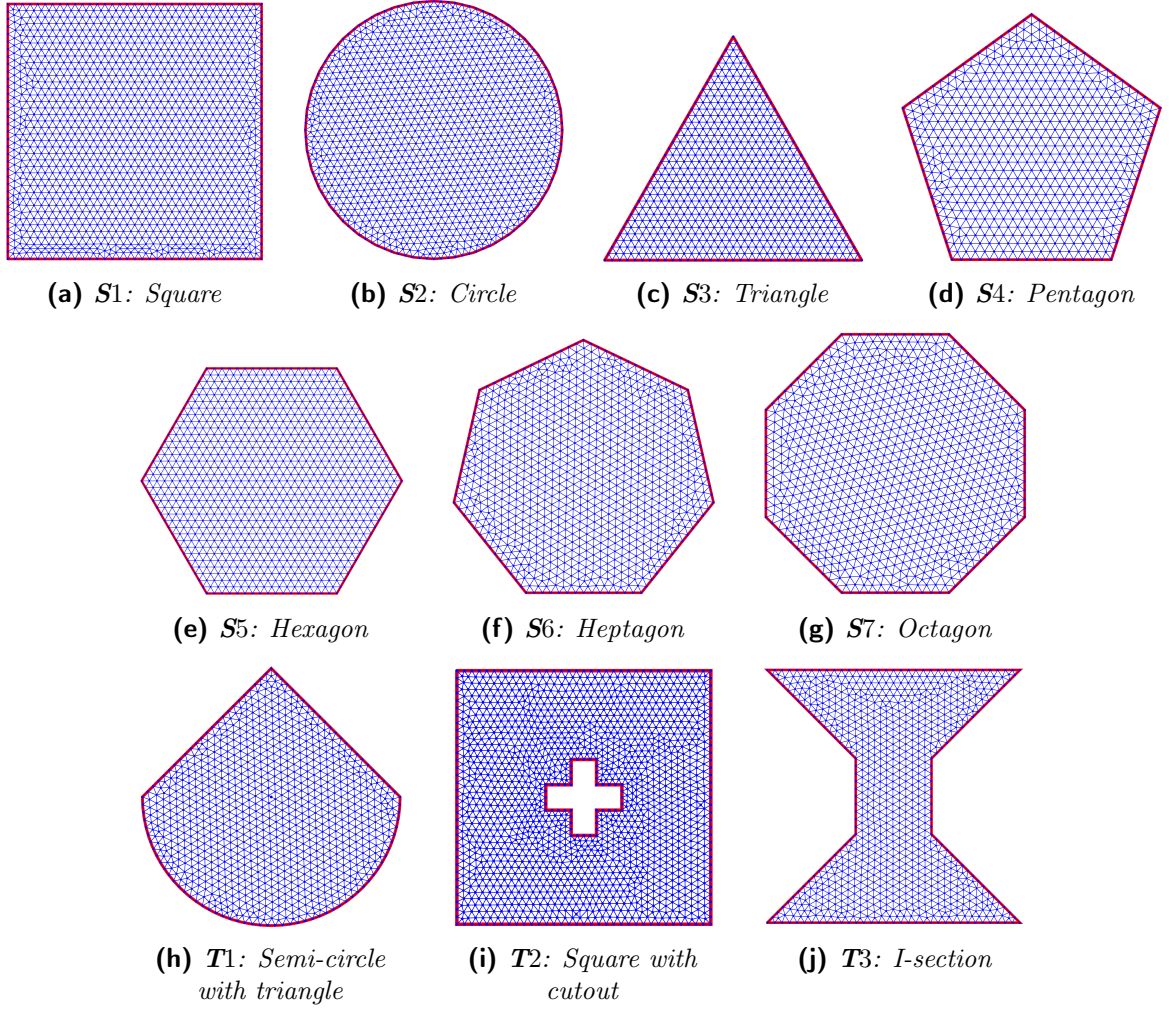
Darcy’s law describes fluid flow through a porous medium, relating pressure, velocity,

and medium permeability. The pressure in the porous medium is expressed as [3]:

$$\nabla \cdot (K(\mathbf{x})\nabla h(\mathbf{x})) = g(\mathbf{x}) \text{ in } \Omega \in \mathbb{R}^2, \quad (2.3.7)$$

$$\text{subject to: } h(\mathbf{x}) = 0, \quad \forall \mathbf{x} \in \partial\Omega, \quad (2.3.8)$$

where  $K(\mathbf{x})$  denotes spatially varying hydraulic conductivity field,  $h(\mathbf{x})$  is the hydraulic head, and  $g(\mathbf{x})$  is the source term. For simplicity, we set  $g(\mathbf{x}) = 1$ . The objectives for multi-task operator learning to predict the hydraulic head,  $h(\mathbf{x})$ , include (i) spatially varying conductivity fields  $K(\mathbf{x})$  drawn from a Gaussian random field, and (ii) operation on varying 2D geometries. In this task, we aim to demonstrate the generalization ability of MT-DeepONet through transfer learning, where the target domains differ geometrically from the source domains. The source and target geometries considered in this example are shown in Figure 2.5. The source geometries are labeled with **S**, while the target geometries are labeled with **T**. The source MT-DeepONet involves training on source domains **S** concurrently with sufficient labeled data, which is later transferred to related target domains **T** where only a small amount of training data is available. To capture geometric variation in a single training session, the conductivity field  $K(\mathbf{x})$  is combined with a binary mask  $\mathbf{M}_{\text{binary}}$  and used as input to the branch network, which employs a CNN architecture. The trunk network receives inputs from a uniformly discretized square domain  $\Omega \in [0, 1] \times [0, 1]$ , subdivided into a  $100 \times 100$  uniform grid. For all geometries, ground truth data is obtained using the MATLAB PDE Toolbox on an irregular mesh and thereafter interpolating the solution onto this  $100 \times 100$  regular grid. The trunk network’s basis functions (output embeddings) are used to evaluate the solution field at all domain points. The solution field outside the domain boundary is enforced to be zero by multiplying the solution operator with the binary mask. This mask ensures the solution is zero outside the geometry, thus improving convergence. Refer to Algorithm 1 showing the implementation details of this workflow.



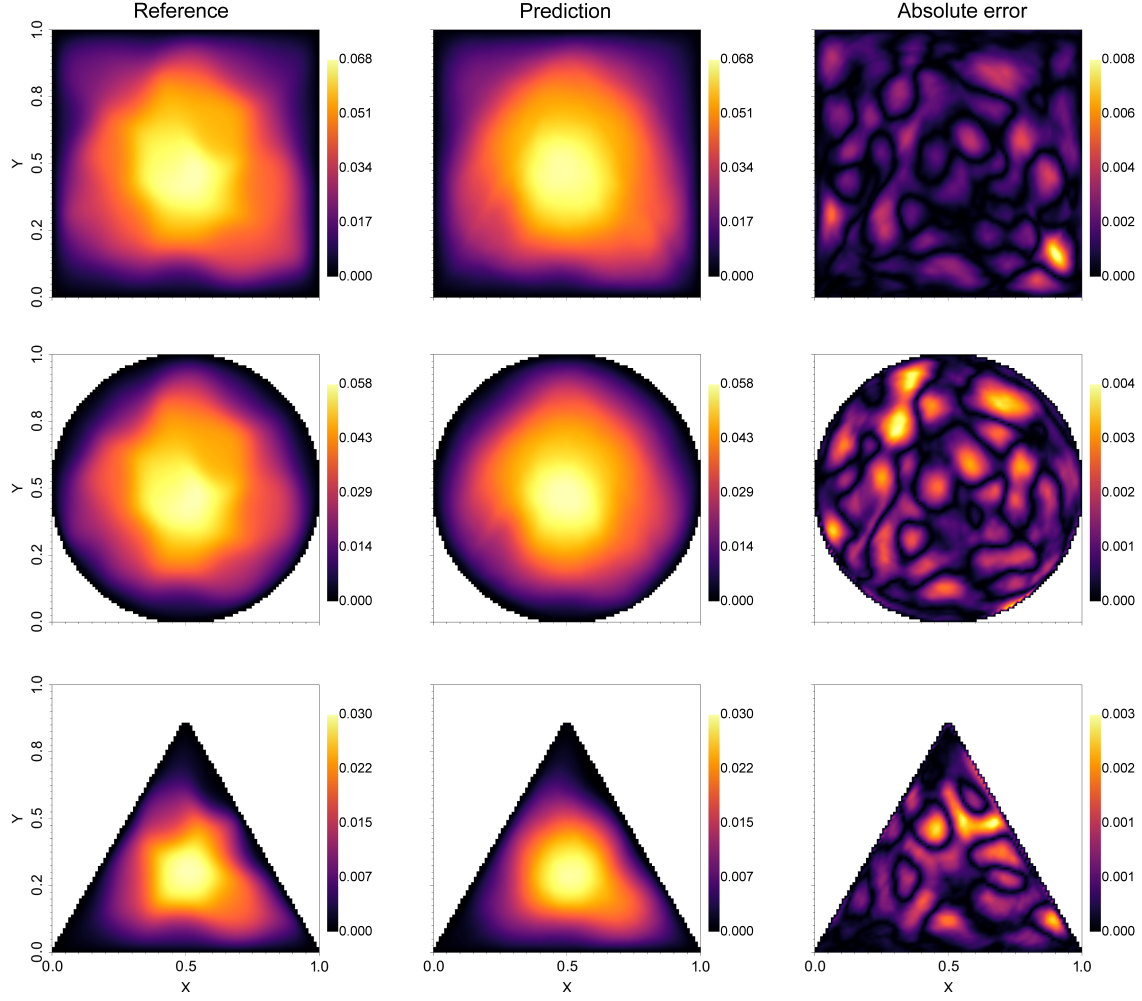
**Figure 2.5:** Different geometries considered as tasks for MT-DeepONet for the Darcy flow problem. The source MT-DeepONet is trained using various combinations of the geometries  $S1$  -  $S7$  while the target domain geometries considered are  $T1$  -  $T3$ . The boundary indicated in red denotes the Dirichlet boundaries where the solution  $h(x) = 0$ . The first objective of MT-DeepONet is to learn the hydraulic pressure heads across a combination of source geometries given a parametric family of spatially varying conductivity fields. The second objective is to transfer the knowledge of source MT-DeepONet to different geometries in the target domain using the transfer learning approach proposed in [1] to reduce overall computation time.

### Transfer learning across different geometries

In this section, we investigate how multi-task training in the source model enhances generalization across different geometries in target models. To assess the effectiveness of transfer learning, we train four source models using various geometric combinations: **S1**, **S1+S2**, **S1+S3**, and **S1+S2+S3**. Each source model is trained on a total of 5,400 samples. For single-geometry models (e.g., **S1**), all samples are from the same

geometry. For multi-geometry models, we evenly distribute samples across the geometries: 2,700 samples from each geometry for **S1+S2** and **S1+S3**, and 1,800 samples from each geometry for **S1+S2+S3**. We train the MT-DeepONet using a piece-wise constant learning rate scheduler with rates  $[0.001, 0.0005, 0.0001]$  over 5,000 epochs, employing mini-batching with a batch size of 1,000 and optimizing with mean squared error. This experimental setup allows us to evaluate how the diversity of geometries in the source model affects the transfer learning process and subsequent performance on target models. Figure 2.6 presents three representative cases of predicted solutions when source MT-DeepONet was trained concurrently with geometries **S1 + S2 + S3**. The binary mask is applied to the output of the operator network to enforce that solutions outside the geometric domain are zero. The results demonstrate that the source MT-DeepONet has accurately learned all three geometries. Table 2.2 presents the  $\mathcal{L}_2$  relative errors for various source model configurations. We observe a trend of increasing error as the number of geometries in the training data grows. This pattern is consistent with our expectations, given the multi-task feature set that the operator network must learn during multi-geometry training. While this approach may lead to a slight reduction in accuracy for individual tasks, it offers a more versatile representation across multiple geometries. This trade-off between task-specific performance and cross-geometry generalization is a key aspect of our multi-task learning strategy.

We employ transfer learning to adapt our trained source multi-task operator network for new geometries **T1** and **T2**. The target models are considered distinct tasks. The transfer learning process begins by initializing the network with trained parameters from the source model. The layers updated during fine-tuning include the first input CNN layer of the branch network, three MLP layers following the convolution modules in the branch, and the linear output layer of the trunk network. We assess the prediction accuracy of the target model using varying numbers of training samples from the target domain: 50, 100, 200, 500, and 800. This approach significantly reduces computational costs for learning pressure heads on new geometries. Table 2.2 presents a summary of errors obtained from



**Figure 2.6:** Representative prediction results for source model trained on geometries **S1** + **S2** + **S3**, concurrently. The MT-DeepONet source model captures the overall features of the solution space across all three geometries reasonably well, making it suitable for use in the transfer learning process.

the target models for different test cases. Our analysis reveals distinct error patterns for target geometries **T1** and **T2**. For **T1**, the source model combination **S1** + **S2** yields a lower  $\mathcal{L}_2$  error compared to the single-geometry **S1** source model. This demonstrates that training on multiple geometries can enhance the network’s transfer learning capabilities. Conversely, source models with geometrical combinations **S1** + **S3** and **S1** + **S2** + **S3** result in higher errors in the target model compared to single-task learning on geometry **S1**. These findings underscore a crucial insight: naively combining all source tasks does not universally improve prediction performance for the target task. This phenomenon, known as negative transfer, occurs when source tasks unrelated to the target tasks are

included. In our case, the results indicate that source geometry **S3** introduces a negative transfer effect. Negative interference is a known challenge in MTL and we plan to explore mitigation strategies in future research. Conversely, for geometry **T2**, the source model **S1** yields similar error values to the combination **S1 + S2**, indicating no significant improvement with the multi-task source model. This result may be attributed to the similarity between geometry **T2** and **S1**. Figures 2.7 and 2.8 compare predictions from the transfer learning model against reference solutions for different source models trained with 50 samples used to fine-tune the target model. The source model was trained with 5400 samples where each geometry had equal representation.

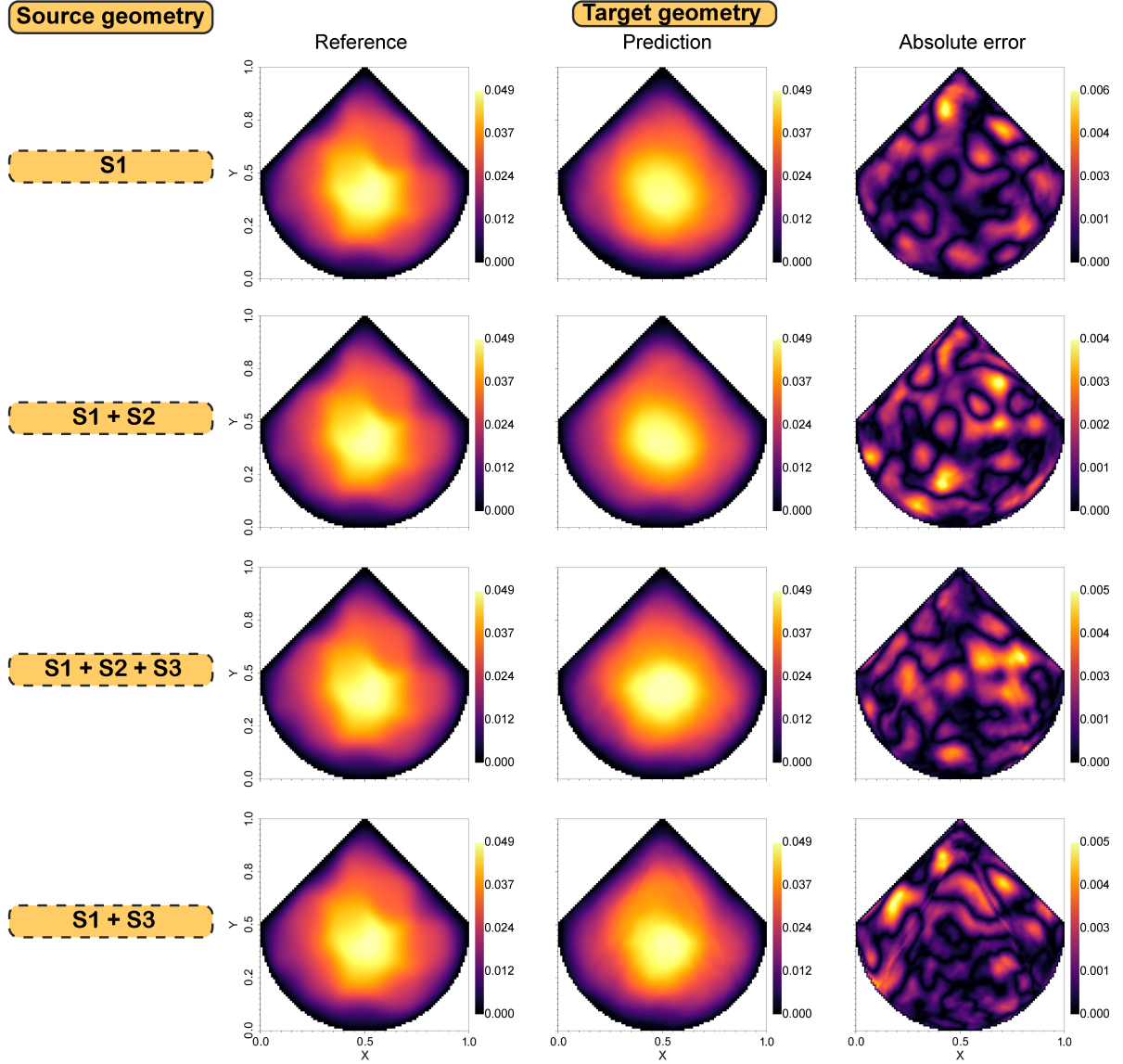
**Table 2.2:** Relative  $\mathcal{L}_2$  error for TL-DeepONet for different source models vs number of samples used during fine-tuning of the target domains, **T1** and **T2**.

| Source geometry model | Source model $\mathcal{L}_2$ rel error | Target geometry | $\mathcal{L}_2$ rel error for target sample size, n |         |         |         |         |
|-----------------------|----------------------------------------|-----------------|-----------------------------------------------------|---------|---------|---------|---------|
|                       |                                        |                 | n = 50                                              | n = 100 | n = 200 | n = 500 | n = 800 |
| <b>S1</b>             | 0.029                                  | <b>T1</b>       | 0.082                                               | 0.063   | 0.055   | 0.047   | 0.045   |
| <b>S1+S2</b>          | 0.039                                  | <b>T1</b>       | 0.065                                               | 0.054   | 0.048   | 0.047   | 0.044   |
| <b>S1+S3</b>          | 0.061                                  | <b>T1</b>       | 0.088                                               | 0.080   | 0.070   | 0.064   | 0.061   |
| <b>S1+S2+S3</b>       | 0.054                                  | <b>T1</b>       | 0.083                                               | 0.068   | 0.060   | 0.055   | 0.054   |
| <b>S1</b>             | 0.029                                  | <b>T2</b>       | 0.158                                               | 0.132   | 0.106   | 0.083   | 0.077   |
| <b>S1+S2</b>          | 0.039                                  | <b>T2</b>       | 0.163                                               | 0.137   | 0.105   | 0.085   | 0.080   |
| <b>S1+S3</b>          | 0.061                                  | <b>T2</b>       | 0.152                                               | 0.134   | 0.117   | 0.102   | 0.098   |
| <b>S1+S2+S3</b>       | 0.054                                  | <b>T2</b>       | 0.158                                               | 0.138   | 0.119   | 0.098   | 0.094   |

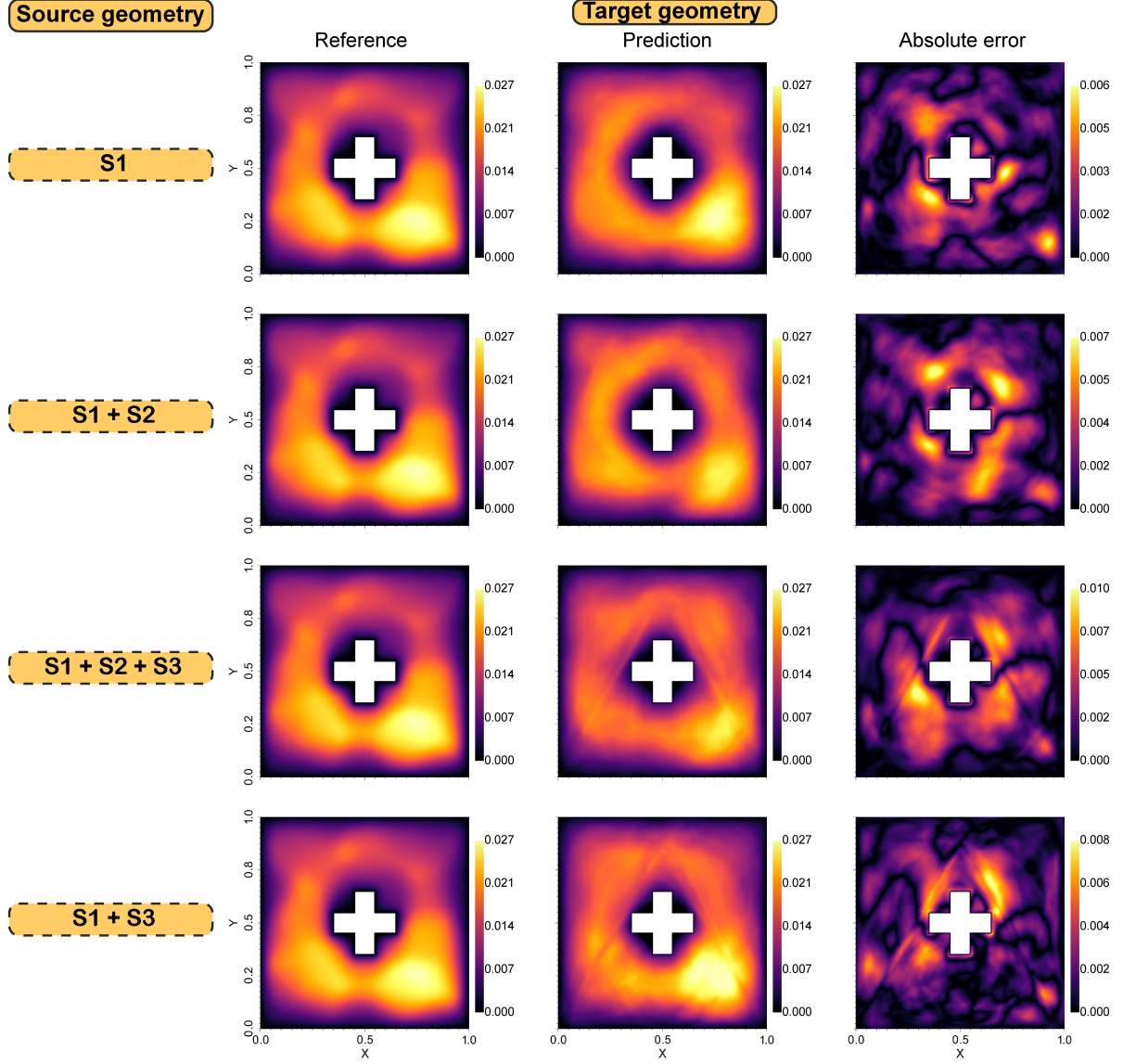
### Transfer learning in the context of disparate geometric domain, **T3**

Our next objective is to investigate whether incorporating multiple geometries in the source model helps reduce the prediction error when transferring to target geometries with different external boundaries. To further explore the efficacy of multi-geometry training in transfer learning, we introduce an I-section geometry (**T3**) as our target for transfer learning. Our hypothesis posits that a source model trained on multiple geometries simultaneously can more effectively transfer knowledge to a new target domain. To validate this hypothesis, we compare the prediction errors of transfer learning models derived from single-geometry source models against the one trained on multiple geometries.





**Figure 2.7:** Comparative analysis between reference solution and prediction generated by the target model  $T1$  trained with 50 samples. Source model  $S1 + S2$  results in lower error values than source model  $S1$  for this geometry. Other source models using  $S3$  in training samples lead to higher errors due to negative interference. Note the artifacts that emerge due to negative interference from the triangular source geometry for source model combinations  $(S1 + S2 + S3)$  and  $(S1 + S3)$ . Overall, the target model is capable of capturing the pressure distribution for new geometries relatively well with the transfer learning methodology when appropriate geometries are chosen in the source model.

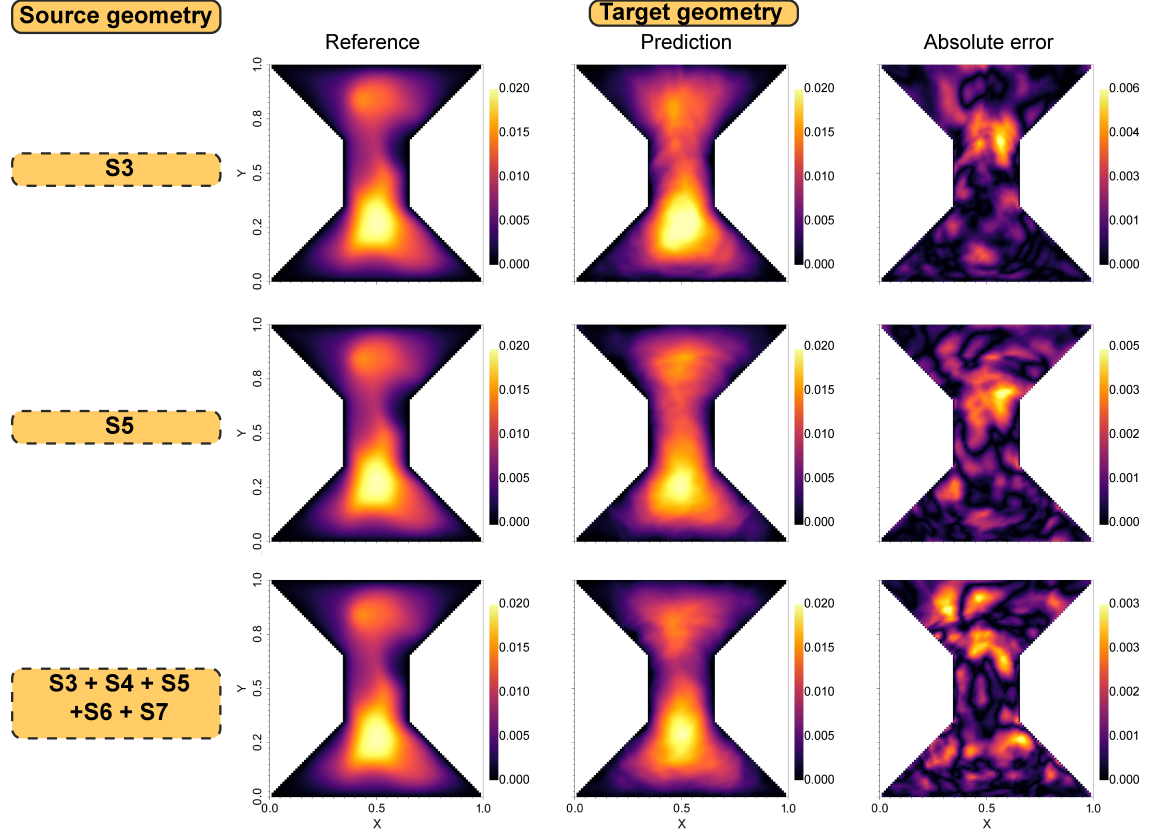


**Figure 2.8:** Comparative analysis between reference solution and prediction generated by the target model  $T2$  fine-tuned with 50 samples. The absolute error plot unveils certain regions with higher error accumulation, especially surrounding the cutout in a square geometric domain ( $T2$ ). Target geometry  $T2$  represents an extrapolated case since none of the source models have an internal Dirichlet boundary with zero pressure head enforced.

Table 2.3 outlines the source geometry combinations employed in this study: **S3**, **S5**, **S7**, and the multi-geometry combination **S3-S7**. For consistency, each source model was trained on a total of 5,400 samples. The results, also presented in Table 2.3, reveal that the lowest prediction error for target geometry **T3** is achieved when the source model is trained on multiple geometries simultaneously. Notably, the improvement in prediction accuracy is more pronounced when the number of target domain training samples is low, with the performance gap narrowing as the number of target domain samples increases. Figure 2.9 provides a visual comparison of MT-DeepONet predictions against reference solutions for target geometry **T3**, using three distinct source geometry configurations. While all models demonstrate the ability to capture high-level features, a marked improvement in model prediction is observed when the source model is trained with multiple geometries. This visual evidence corroborates our quantitative findings and underscores the potential benefits of multi-geometry training in enhancing the transferability and generalization capabilities of our model.

### **Transfer learning with fine tuning of additional trunk layer**

In previous discussions, we presented findings on transfer learning across different 2D domains, focusing on fine-tuning the last trunk layer along with the first CNN module and the three MLP layers within the branch network. We also evaluated the impact of increasing the number of trainable parameters during the transfer learning phase on the model’s accuracy. Specifically, we trained the last two layers of the trunk network (the output layer and the last non-linear hidden layer). We chose to use an additional trunk layer for fine-tuning based on experiments where we observed better accuracy with an additional trunk layer compared to a CNN layer in a branch network. Empirical evidence from our experiments suggests that retraining the last hidden layer has a marginal impact on computational speed. Table 2.3 provides an overview of the errors observed across various source-target combinations. Compared to fine-tuning with a single trunk layer, transfer learning with two trunk layers shows a marginal improvement in prediction accuracy across all combinations, particularly for cases with larger sample sizes during



**Figure 2.9:** A comparison between the reference solution and the prediction from our multi-task operator network for the transfer learning model, trained with 50 samples on geometry combinations of *S3-S7*. The *I*-section geometry used as the target is significantly different from the geometries used in the source model. Utilizing a source model trained on multiple geometries for transfer learning results in a marginally lower prediction error which may not be significant.

transfer learning. This indicates that using additional trainable parameters during transfer learning does not improve the results significantly.

### 2.3.3 Heat transfer through multiple 3D geometries

The steady-state heat transfer equation for an isotropic medium is defined as:

$$-\nabla \cdot (K \nabla \mathbf{u}) = q, \quad \Omega \in \mathcal{R}^3 \quad (2.3.9)$$

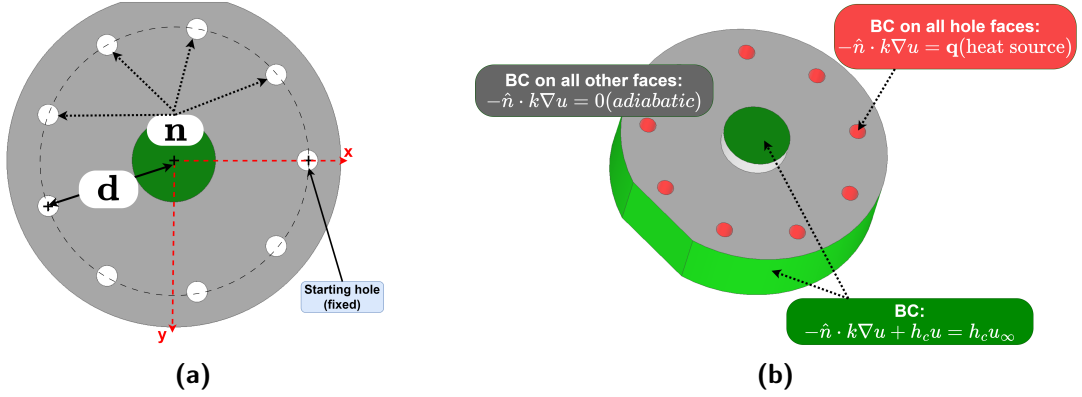
where  $K$  is the thermal conductivity,  $q$  is the internal heat source, and  $\mathbf{u}$  is the spatially varying temperature field. For multi-task operator learning, we solve this equation on a parametric 3D circular plate as outlined in [66]. The plate has an outer diameter of

**Table 2.3:** Error values for different sample sizes used to fine-tune the target model under the transfer learning scheme for the target domain, **T3**. The table presents the results for two transfer learning scenarios, each differing in the layers selected for fine-tuning.

| Source geometry<br>model                                    | Source model<br>$\mathcal{L}_2$ rel error | Target<br>geometry | $\mathcal{L}_2$ rel error for<br>target sample size, n |         |         |         |         |
|-------------------------------------------------------------|-------------------------------------------|--------------------|--------------------------------------------------------|---------|---------|---------|---------|
|                                                             |                                           |                    | n = 50                                                 | n = 100 | n = 200 | n = 500 | n = 800 |
| <i>Results with fine-tuning 3 branch and 1 trunk layer</i>  |                                           |                    |                                                        |         |         |         |         |
| S3                                                          | 0.045                                     | T3                 | 0.177                                                  | 0.163   | 0.142   | 0.126   | 0.120   |
| S4                                                          | 0.042                                     | T3                 | 0.176                                                  | 0.159   | 0.141   | 0.120   | 0.115   |
| S5                                                          | 0.043                                     | T3                 | 0.172                                                  | 0.161   | 0.140   | 0.114   | 0.107   |
| S6                                                          | 0.042                                     | T3                 | 0.187                                                  | 0.164   | 0.153   | 0.129   | 0.116   |
| S7                                                          | 0.042                                     | T3                 | 0.183                                                  | 0.166   | 0.148   | 0.128   | 0.117   |
| S3+S4+S5+S6+S7                                              | 0.063                                     | T3                 | 0.167                                                  | 0.154   | 0.138   | 0.125   | 0.116   |
| <i>Results with fine tuning 3 branch and 2 trunk layers</i> |                                           |                    |                                                        |         |         |         |         |
| S3                                                          | 0.045                                     | T3                 | 0.158                                                  | 0.143   | 0.124   | 0.102   | 0.090   |
| S4                                                          | 0.042                                     | T3                 | 0.165                                                  | 0.143   | 0.118   | 0.097   | 0.088   |
| S5                                                          | 0.043                                     | T3                 | 0.181                                                  | 0.148   | 0.125   | 0.100   | 0.091   |
| S6                                                          | 0.042                                     | T3                 | 0.189                                                  | 0.157   | 0.133   | 0.109   | 0.098   |
| S7                                                          | 0.042                                     | T3                 | 0.181                                                  | 0.153   | 0.141   | 0.111   | 0.101   |
| S3+S4+S5+S6+S7                                              | 0.063                                     | T3                 | 0.161                                                  | 0.142   | 0.128   | 0.104   | 0.096   |

4 inches, a central protrusion of height  $\frac{1}{4}$  inch, and an overall thickness of 1 inch. The location and the number of holes are parameterized as follows: the holes are placed at distances  $\mathbf{d} \in [0.9, 1.6]$  inches from the center of the protrusion, with the number of holes  $\mathbf{n}$  varying between 2 and 9. All holes are equispaced in all the considered geometries, while the location of one hole is fixed in all the geometries in the plate (see Figure 2.10a). The radial location of the holes,  $\mathbf{d}$  is varied in steps of 0.1 inch. Heat sources  $\mathbf{q}$  are placed in each hole, with a convective heat flux on the protrusion and side face, and adiabatic conditions elsewhere. These natural convective boundary conditions are shown in Figure 2.10b. The objective is to learn the temperature distribution across the 64 geometries that result from combinations of  $\mathbf{n}$  and  $\mathbf{d}$  (details of data generation in Section A1.2). Figure 2.11 presents four representative geometries and the corresponding temperature fields obtained from MATLAB's PDE Toolbox.

Our objective here is to learn the operator mapping between the geometry parameters

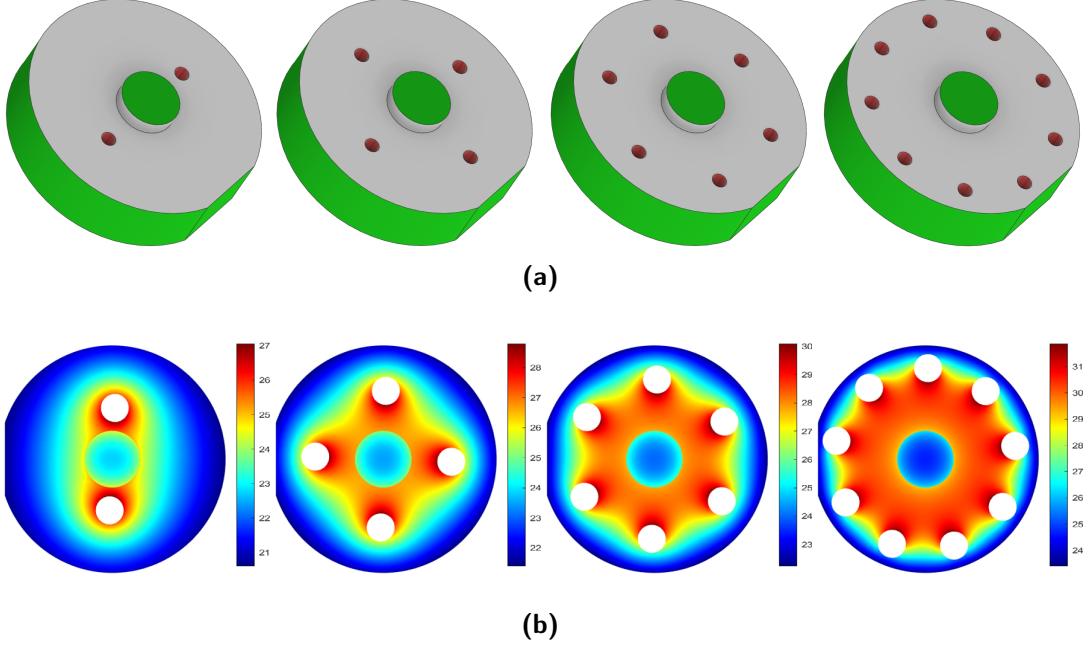


**Figure 2.10:** (a) View of a 3D plate showing the parametric design features, where  $\mathbf{n}$  represents the number of holes and  $\mathbf{d}$  denotes the location of the holes relative to the central axis of the protrusion face. Multiple geometries are generated by varying  $\mathbf{n}$  and  $\mathbf{d}$  within their design ranges. (b) depicts the boundary conditions used for solving the heat transfer equation. The top protrusion and the side wall have a convective heat flux boundary condition, while the holes act as individual heat sources. The solution field consists of the temperature distribution across the 3D plate domain.

defining the different plate configurations and the resulting temperature field due to the heat sources placed inside the holes located in the plate. Since the number and location of the holes change based on the parameters  $n$  and  $d$ , the resulting temperature field varies for each plate configuration. For the operator learning purpose, we select a total of 24 training samples that encompass the minimum, maximum, and median values of the parameter  $d$  for different numbers of holes. For instance, for a plate with  $n = 3$ , we choose three samples with holes located at  $d = 0.9$ ,  $d = 1.2$ , and  $d = 1.6$  as training samples corresponding to  $n = 3$ . The sample choice ensures that the extreme ends of the design space are well represented in the training dataset while avoiding oversampling. The remaining geometries are used as test cases for network inference. A list of the cases used for training and testing is provided in Table A1.

The parametric representation for the plate ( $n$  and  $d$ ) is used as input to our branch network, while the trunk network receives uniformly sampled points in the domain  $\Omega = \{(x, y, z) \in \mathbb{R}^3 : -2 \leq x \leq 2, -2 \leq y \leq 2, -0.25 \leq z \leq 1\}$  as input. Unlike the previous example on the 2D Darcy problem where the binary mask was considered as input to the branch network, here only the geometry parametrization values are employed. The binary mask is used as additional information in the loss function. We utilize the

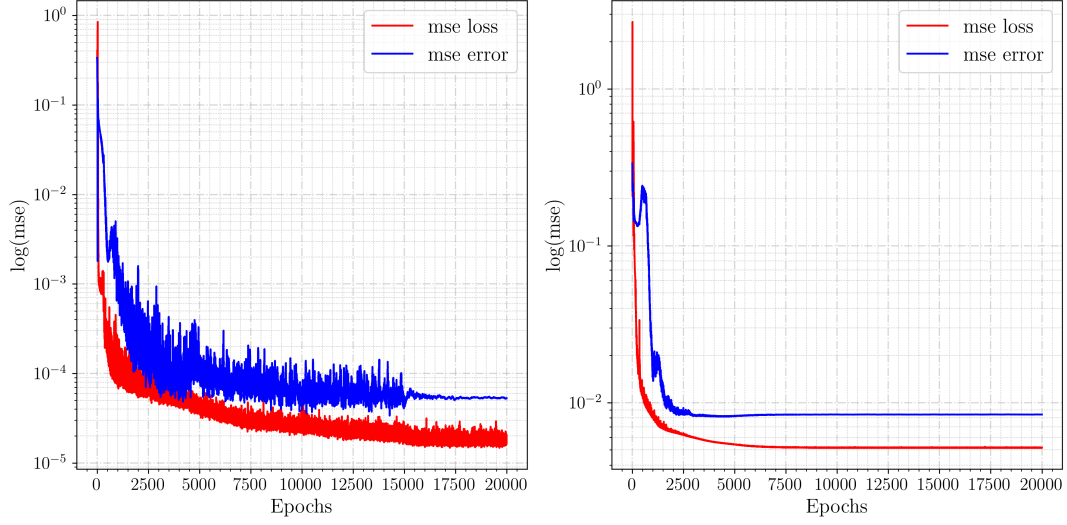




**Figure 2.11:** (a) Representative 3D plate geometry configurations generated by changing the two design parameters:  $\mathbf{n}$  and  $\mathbf{d}$ . (b) Variations in temperature fields for different geometric configurations and number of heating sources. The multi-task objective is to learn the mapping from geometric parameters to temperature fields for different geometric configurations.

same grid points for all plate configurations with the trunk network and later apply a binary mask to constrain the temperature field to be zero outside the geometry, similar to the 2D Darcy problem. The masking matrix is a 3D grid with values of 0 (outside) and 1 (inside) the domain boundary. By utilizing this binary mask, we establish the associativity between the grid points and geometry points, enabling the operator network to learn the temperature field across multiple geometries simultaneously. The training step employs an exponentially decaying learning rate, starting at  $1 \times 10^{-3}$  with a decay rate of 0.9 every 1000 iterations.

The binary mask, applied to the network output, ensures that the solution at points outside the geometry is zero, thereby improving convergence and accuracy. Figure 2.12 compares the convergence of the MT-DeepONet with and without the masking function in the loss function. The application of a mask to the solution output demonstrates better error convergence. Figure 2.13 shows the prediction of MT-DeepONet without the masking operation, where higher inaccuracies are observed across the domain due to insufficient geometry representation in the training dataset. Applying a binary mask imposes an



**Figure 2.12:** Comparison of convergence rates between a network trained with the loss function as a product of the solution operator and the masking function (left) and one trained without a masking operation in the loss function (right). The application of the mask to the solution output demonstrates better error convergence and a reduction in generalization error.

extra constraint that assists the network in learning solutions across different geometries. This masking process supplies essential geometric details and improves the network’s capacity to generalize to new plate configurations (with the same  $n$ ). Figure 2.14 presents representative examples of geometries comparing the reference temperature field with the network prediction. These predictions align well with the reference numerical solution. Higher error values are observed near the holes due to their being under-represented in the training data. We observed better results when more geometries are added to the training dataset but this leads to oversampling in the training dataset. The masking framework functions similarly to transfer learning between tasks, albeit without involving network re-training and is critical for predicting solutions across different geometries in our experiments.

## 2.4 Limitations and future work

Through our experiments, we have demonstrated the usefulness of a multi-task deep operator network as a step towards creating generalized surrogates for PDE solvers with



machine learning. While learning common representations across multiple tasks enables the neural operator to improve generalization and knowledge transfer to new tasks, there is a possibility of negative interference due to tasks that appear related but are in fact not relevant. Perhaps, a good research opportunity is to explore how individual tasks relate to each other before using them together in a multi-task framework. In our 2D Darcy flow problem, we do see the impact of negative interference due to using more than two geometries. Works by [37] and [67] provide unique concepts for managing information sharing between layers of the neural network for image processing applications. As future work, we will explore the possibility of extending these ideas for optimizing the knowledge sharing between tasks for scientific problems, such as the ones we presented here.

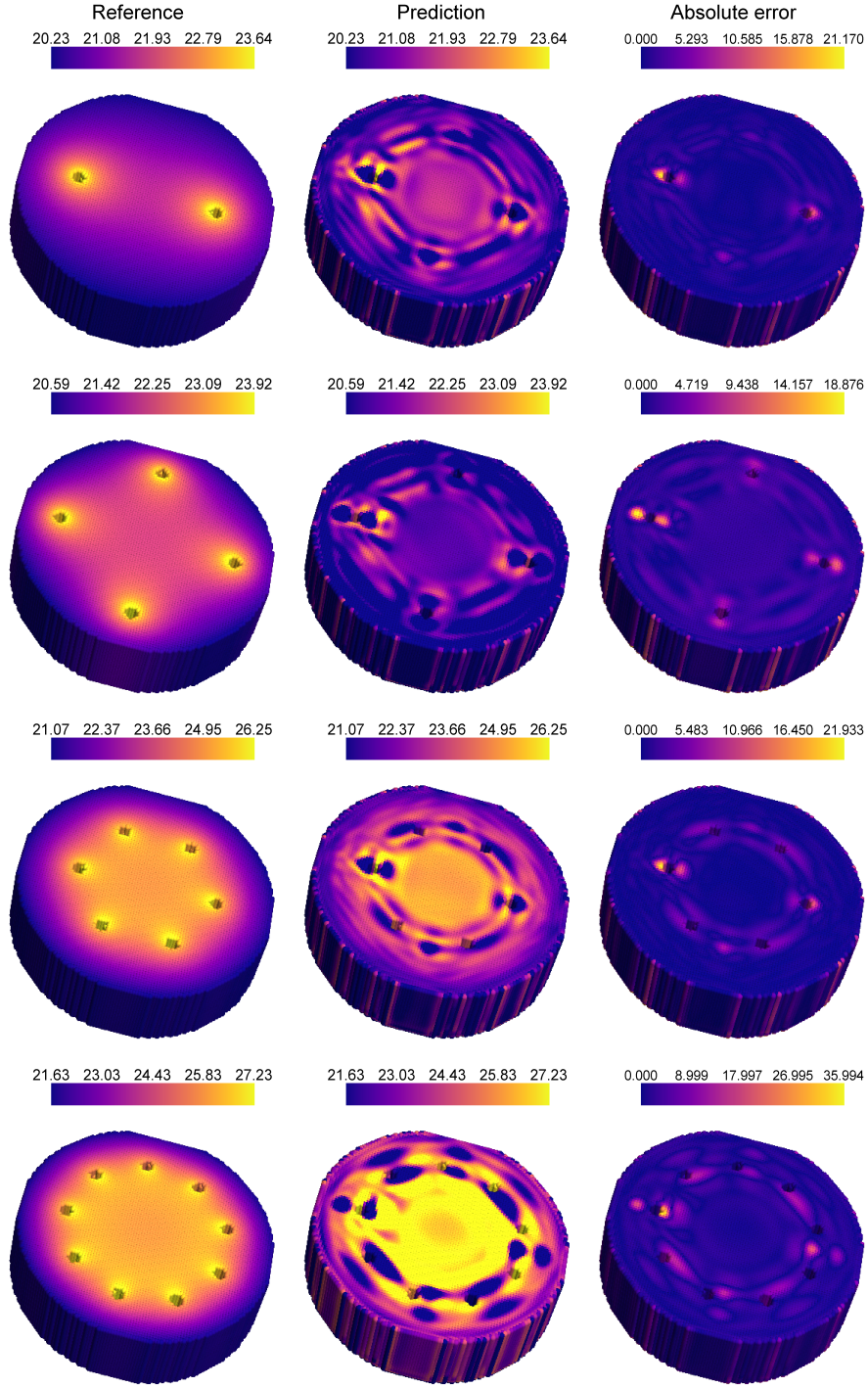
A critical limitation of this method is the need to interpolate the solutions for different geometry onto a common grid for DeepONet training. This is essential since the DeepONet framework requires a bounded domain to learn the operator mapping between functional spaces. When the domain used for training changes for each geometry, a much larger sample size is required to generalize a unique functional-to-functional operator mapping. Since the sample size is often limited for scientific and engineering applications, we compensate the need for large sample size by training the network on an interpolated solution on a uniform grid and later creating a unique representation for each geometry through the masking operation. Creating such regular grids may not be possible for some realistic geometries, such as surface-based geometries, e.g. thin shells.

Another research opportunity that we identify in this work is the need for generating a parametric representation of complex 3D geometries relevant to engineering applications, such as the 3D plate example we present here. In this case, the differentiating parameters between two different plate geometries are the number of holes and their location, and hence, we use this parametric representation as an input to our branch network. In applications where this parametric representation is unknown, geometric encoding information is required to differentiate between each sample. One of our key research objectives in the future is to develop neural representations for encoding 3D geometries relevant to

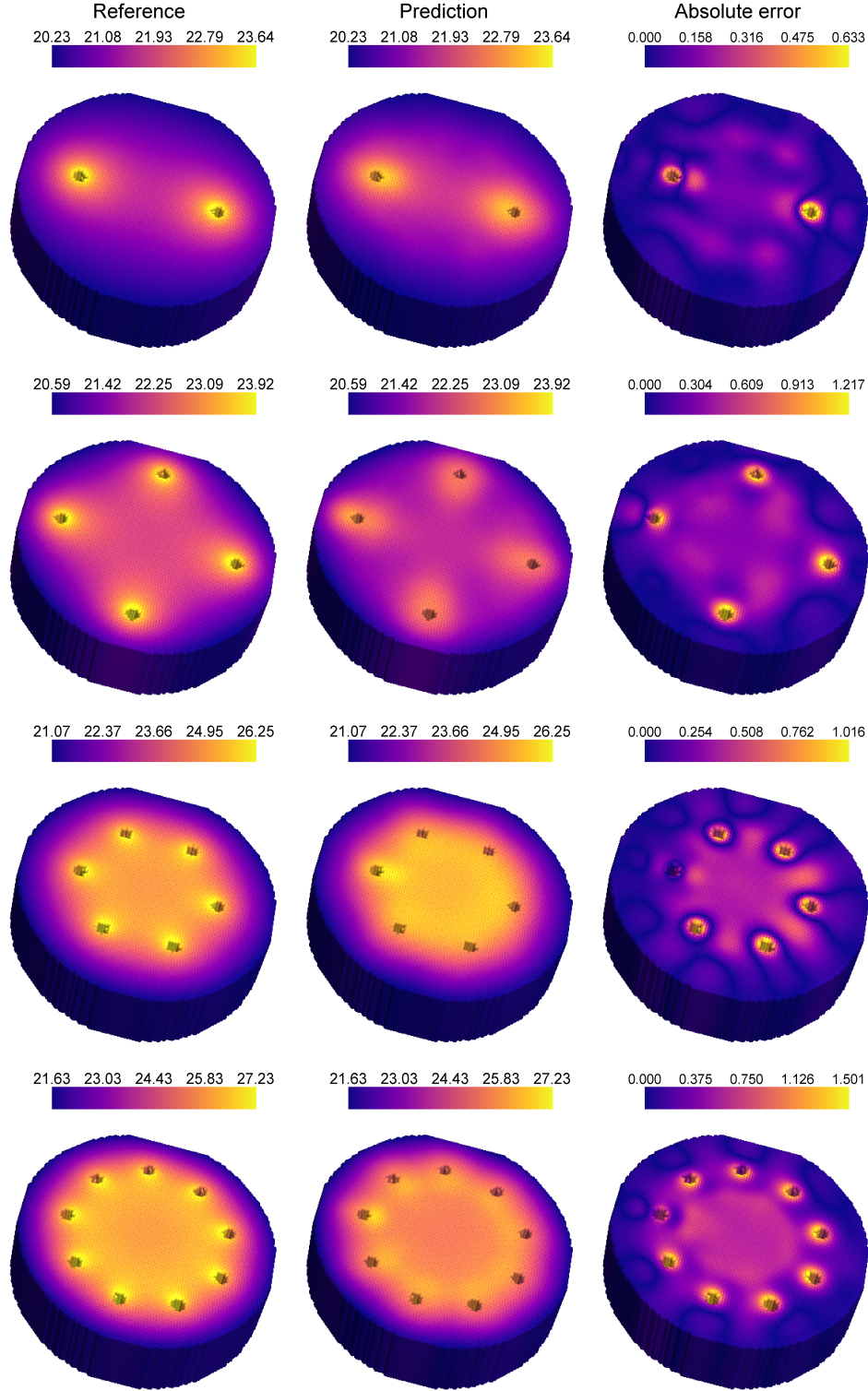
engineering and scientific work. We can draw inspiration from the works of [68, 69, 70] to extend these ideas to engineering shapes and geometries.

## 2.5 Summary

In this work, we introduced two frameworks for multi-task learning that leverage the DeepONet architecture (MT-DeepONet). These frameworks are designed to enable efficient learning across diverse scenarios, encompassing variations in geometries and physical systems, within a single training session. While the architectures are tailored for specific applications, their versatility is evident through their ability to handle a wide range of physical processes. The first framework, which employs a parametrized representation, is demonstrated using the Fisher equation, while the second framework, utilizing a binary mask, effectively captures varying geometries in applications such as Darcy flow and heat transfer. The MT-DeepONet framework shows a strong ability to map operator solutions across multiple physical processes as well as multiple geometries, and showcasing capabilities in transferring knowledge across different PDE forms and geometric configurations. While achieving competitive accuracy in predicting solutions, challenges such as managing negative interference between tasks and generalizing to unseen geometries were evident, particularly in the context of learning multiple varied geometries. The binary mask function introduced in the loss term imposes the boundary condition on different geometries as well as improves convergence and generalization. Future research directions should include optimizing network architectures to enhance transfer learning efficacy, developing robust methods for encoding complex 3D geometries, and exploring advanced techniques to mitigate negative inference in MT-DeepONet.



**Figure 2.13:** Four representative geometries with different numbers and locations of the holes to compare the accuracy of the MT-DeepONet solution (without the masking operation) with the reference solution. The results revealed higher errors across the plate compared to the masked solution (see Figure 2.14). Applying the binary mask imposes an additional constraint that assists the network in learning solutions across varying geometries, particularly when inferring new geometries not seen during training. The masking operation provides crucial geometric information, enhancing the network's ability to generalize to unseen configurations and thereby reducing overall errors.



**Figure 2.14:** Comparison between reference temperature field solution and predictions generated by the MT-DeepONet, for four representative geometries parameterized by number and location of the holes. The prediction shows good agreement with the reference across the domain except for some regions around the holes. We used the parametric representation for the target geometry by providing  $\{n, p\}$  as an input to the branch network for this study and augmented the low-fidelity input data by applying the masking operation on the solution.

# CHAPTER 3

## Fusion-DeepONet: A Data-Efficient Neural Operator for Geometry-Dependent Hypersonic and Supersonic Flows

### 3.1 Introduction

High-speed flows involve complex interactions between shock waves, boundary layers, chemical reactions, and thermal effects, which vary significantly with the vehicle's geometry. These interactions require large-scale, high-fidelity, and computationally expensive simulations, particularly when exploring multiple geometric configurations. Given this, running full simulations for every potential design is computationally intractable. Multiple design parameters (e.g., shape, material properties, surface roughness) must be tested in hypersonic and supersonic vehicle design to find the optimal configuration. Geometry-dependent surrogate models can efficiently explore various geometries, helping engineers optimize designs without running costly CFD simulations. Moreover, real-time feedback is often necessary in scenarios such as launch vehicle guidance or rapid prototyping. Surrogate

models can offer fast predictions during these operational stages, enabling decision-making on-the-fly based on geometric changes or modifications in flow conditions. Surrogate models based on multi-fidelity setup have traditionally been used for modeling flow around objects but have primarily focused on sub-sonic regimes [71, 72, 73, 74, 75, 76, 77]. Recently, neural networks have emerged as an important class of surrogate models for inferring flow fields. With their ability to learn complex non-linear mapping between input and output functions, neural networks provide a promising landscape for surrogate model development to replace expensive CFD simulations. The research impetus on neural network-based surrogate models for aerodynamic flows has led to a surge in deep learning-based surrogates [78, 79, 80, 81, 82, 83, 84]. However, these existing methods mainly focus on sub-sonic flows, likely due to the challenges of solving hypersonic problems that typically involve extreme variations in flow fields in space and time.

An important limitation of traditional neural network-based methods for scientific tasks is the lack of discretization invariance property. The discretization invariance is important since the solution space often requires interpolation over the entire domain rather than being limited to pre-defined grid points. As an alternative, operator networks offer the unique property of discretization invariance in temporal or spatial domains, allowing for better generalization capabilities over Vanilla neural networks. Some prominent operator-based frameworks include Deep Operator Network (DeepONet) [85], Fourier Neural Operator (FNO) [30], Laplacian Neural Operator (LNO) [86], Wavelet Neural Operator (WNO) [87], to name a few. Operator-based methods have seen significant growth in their use as surrogate models for scientific tasks that require significant computational overhead with traditional numerical tools. Operator networks have also been explored for aerodynamic modeling, and some significant works include [4, 88, 89, 90]. While the application of these operator methods has been explored for aerodynamic problems such as flow around airfoils, their use has largely remained unexplored for predicting flow around hypersonic bodies, especially when the shape of the hypersonic body is changing. Hence, a research opportunity exists to determine the effectiveness of these

methods for geometry-dependent hypersonic flow problems and further develop a general operator-based framework to tackle such tasks.

Prediction of hypersonic and supersonic flows is challenging due to extremely high-gradient fields around the flying vehicle. For instance, the formation of the bow shock in front of blunt bodies creates a high-temperature area between the nose and the shock. High temperature reduces the density and creates near-vacuum conditions in some locations. Therefore, the data generation for hypersonic and supersonic flows is challenging; hence, only a few high-fidelity numerical solvers exist. Recently, Peyvan *et al.* [91] developed a high-order entropy stable scheme that can robustly simulate supersonic to hypersonic flows and provide high-fidelity data. Due to the scarcity of high-speed flow data and the complications involved with such simulations, limited studies focus on developing surrogate models for such flows. For instance, Way *et al.* [92] created a surrogate model of hypersonic flows that maps from free stream flow conditions, including Mach number, angle of attack, and altitude, to the temperature and shear stresses on the surface of a hypersonic vehicle. However, their framework cannot handle a geometry-dependent flow field. In another case, Rataczak *et al.* [93] created surrogate models for hypersonic aerodynamics using Gaussian process regression. They created three surrogate models. The first surrogate predicts the stagnation point heat flux and axial force coefficients. The second model predicts convective heat flux contours on the vehicle surface. Finally, the third surrogate is designed to predict the vehicle lift and drag coefficient by varying the angle of attack. Scherding *et al.* [94] employed reduced order nonlinear approximation with adaptive learning procedure to train a surrogate model for the thermodynamics in CFD solvers with application to unsteady hypersonic flows in chemical non-equilibrium. Schouler *et al.* [95] trained Krigging and fully connected neural networks to predict the pressure, heat flux stagnation coefficient, friction, and heat flux coefficient distributions in the rarefied portion of a reentry vehicle. The previous hypersonic and supersonic surrogate modeling lacks the consideration of the flying object geometry variation and learning of the flow field.



Learning geometry-dependent PDE solutions is essential in the geometric design of physical systems. In that category, Zhao *et al.* [96] developed a Neural Operator for shape-dependent PDE solutions by mapping any physical domain to a generic reference domain using a diffeomorphic transformation. Then they defined parameter and solution functions in this generic domain and let an improved Fourier Neural Operator (FNO) learn the mapping from parameter functions to the PDE solution field in the reference domain. Although they showed a promising prediction precision for various physics, the framework has several deficiencies. For instance, for any new geometry, there must be a generic reference domain. Creating a reference domain for complex geometry is a significant challenge. The framework does not work if we cannot derive an invertible mapping function that transforms the physical domain into the reference domain. Yin *et al.* [97] also developed Diffeomorphic Mapping Operator Learning (DIMON) that learns geometry-dependent solutions of PDEs using a multi-branch architecture of DeepONet. They again map the physical domain onto a reference domain and reformulate the PDEs on the reference geometry. The neural operator then learns the PDE solution on the reference geometry. Casenave *et al.* [98, 99] also employed a mesh morphing technique to transform complex geometries with the same topology into a common mesh, then used a classical dimension reduction (PCA) and Gaussian processes to construct a surrogate for the non-parametric geometries. Catalani [100] also used the INR network to encode the geometry and use the encoded geometry as the input parameter in a conditioned neural field. Helwig *et al.* [101] employed a message passing framework to enable the neural network to learn the geometry of airfoils, thereby allowing it to learn the subsonic flow field around them using a graph neural network.

In high-speed flow simulations, key quantities of interest typically include shear stress and heat flux acting on the surfaces of flying vehicles. Accurate prediction of these quantities necessitates precise computations of spatial gradients in the solution fields. Neural operators are conventionally trained by minimizing the mean squared error between predicted and ground truth solution fields. However, this approach lacks explicit



consideration of derivative information in the target data. A few recent studies have addressed this gap. Cao et al. [102] proposed a derivative-informed neural operator (DINO) framework, which trains surrogate models for costly parameter-to-observable mappings by simultaneously minimizing both output errors and Jacobian errors through a Sobolev loss. This method significantly reduces the required number of iterations during minimization of a cost function in an optimization loop. In the context of DeepONet, the Jacobian is calculated with respect to the network inputs. Similarly, Qiu et al. [103] extended the DINO concept by developing a derivative-enhanced Deep Operator Network, incorporating linear dimensionality reduction for high-dimensional parameter inputs and a derivative-based loss term. This enhancement improves both solution accuracy and the precision of solution-to-parameter derivatives, particularly in scenarios with limited training data. In our current study, we focus specifically on the derivative with respect to spatial coordinates. We identify the optimal form of derivative-informed loss to accurately predict quantities like heat flux and determine the derivative operator most compatible with high-fidelity CFD datasets.

Specifically, we develop a new Fusion-DeepONet framework and compare its performance with other neural operators to map geometry to the hypersonic flow field. We test Fusion-DeepONet to learn three geometry-dependent problems, including inviscid hypersonic flow over semi-ellipse bodies, viscous hypersonic flow around a reentry capsule, and internal supersonic compressible flow inside a converging-diverging nozzle. Temperature predictions are used to compute the heat flux exerted on the surfaces of reentry capsules. We summarize the main contributions of our work below:

- We develop Fusion-DeepONet, a geometry-aware neural operator that accurately predicts hypersonic flow fields on uniform and irregular grids using scarce data. It achieves comparable accuracy to parameter-conditioned U-Net for uniform grid while outperforming MeshGraphNet and Vanilla-DeepONet for irregular grids. With the lowest number of trainable parameters among the models studied, Fusion-DeepONet

ensures computational efficiency, scalability, and applicability to complex geometries.

- The singular value decomposition (SVD) of the trunk network reveals that Fusion-DeepONet learns both high- and low-frequency features of the hypersonic solution better than the vanilla model.
- We demonstrate that Fusion-DeepONet accurately infers internal and external viscous high-speed flows across geometric configurations.
- We introduce a derivative-enhanced loss function that enables Fusion-DeepONet to predict the temperature field around a reentry capsule with sufficient accuracy to yield accurate heat flux estimates on its surface.

The rest of the article is arranged as follows: We describe the governing equations solved for the data generation. Next, we explain the data generation procedure for training and testing the neural networks. The methodology section describes the structure of Fusion-DeepONet and the loss functions used for training. We then illustrate and interpret the results. Finally, we conclude the paper with a summary. The necessary information and data for reproducibility are included in the appendix section.

## 3.2 Governing Equations

We are interested in learning geometry-dependent flow fields generated by solving the 2D compressible Navier-Stokes equations given by

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}^a}{\partial x} + \frac{\partial \mathbf{G}^a}{\partial y} = \frac{1}{Re} \left[ \frac{\partial \mathbf{F}^v}{\partial x} + \frac{\partial \mathbf{G}^v}{\partial y} \right] \quad (3.2.1)$$

where the vector of conserved variables  $\mathbf{U}$  is defined as

$$\mathbf{U} = (\rho, \rho u, \rho v, \rho E)^T, \quad (3.2.2)$$

and convective flux vectors  $\mathbf{F}$  in x-direction and  $\mathbf{G}$  in y-direction are:

$$\mathbf{F}^a = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ u(\rho E + p) \end{pmatrix}, \quad \mathbf{G}^a = \begin{pmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ v(\rho E + p) \end{pmatrix}. \quad (3.2.3)$$

The viscous fluxes are expressed as

$$\mathbf{F}^v = \begin{pmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ u\tau_{xx} + v\tau_{xy} - q_x \end{pmatrix}, \quad \mathbf{G}^v = \begin{pmatrix} 0 \\ \tau_{xy} \\ \tau_{yy} \\ u\tau_{xy} + v\tau_{yy} - q_y \end{pmatrix}. \quad (3.2.4)$$

In Eq. (3.2.2),  $\rho$  denotes density. The terms  $u$  and  $v$  represent velocity components along x and y directions respectively.  $\rho E$  indicates the total energy of the gas mixture, including internal and kinetic energies defined as

$$\rho E = \frac{p}{\gamma - 1} + \frac{1}{2}\rho(u^2 + v^2), \quad (3.2.5)$$

where  $\gamma$  is the ratio of specific heat capacity for fluid medium with value  $\gamma = 1.4$  for air. In Eq.(3.2.5),  $p$  denotes pressure. The Newtonian stress tensor using the Stokes's hypothesis is:

$$\begin{aligned}\tau_{xx} &= 2\mu \frac{\partial u}{\partial x} - \frac{2}{3}\mu \left( \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right), & \tau_{yy} &= 2\mu \frac{\partial v}{\partial y} - \frac{2}{3}\mu \left( \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right), \\ \tau_{xy} &= \mu \left( \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right),\end{aligned}\tag{3.2.6}$$

where  $\mu$  is the constant non-dimensional dynamic viscosity. According to Fourier's law of heat conduction, the heat fluxes in Eq.(3.2.4) take the following form:

$$q_x = -\kappa \frac{\partial T}{\partial x}, \quad q_y = -\kappa \frac{\partial T}{\partial y},\tag{3.2.7}$$

where  $\kappa$  is defined as  $\kappa = \frac{\mu}{Pr} \frac{\gamma}{\gamma - 1}$ .  $Pr$  denotes the Prandtl number and  $Re$  refers to reference Reynolds number. In Eq (3.2.7),  $T$  refers to the non-dimensional temperature. For non-dimensionalization, we have set the reference values of quantities as the free stream value. In detail, we set the reference velocity equal to the speed of sound computed using the free stream condition. The reference temperature is also computed using the ideal gas law with pressure and density set as free stream conditions and gas constant equal to  $R = 287.058$  JKg/K. The non-dimensional temperature follows the ideal gas law as

$$p = \gamma \rho T.\tag{3.2.8}$$

### 3.3 Data Generation

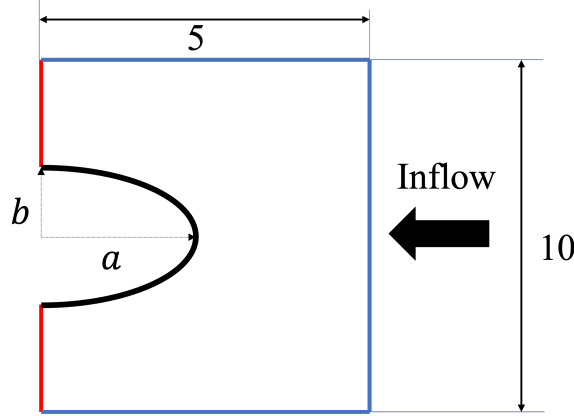
In this study, we intend to learn high-speed flows for three geometry-dependent problems, including inviscid hypersonic flow around semi-ellipses, viscous supersonic flow inside a converging-diverging nozzle attached to an isolator duct, and a reentry capsule traveling at hypersonic speed. All problems involve samples with varying geometries. We employ the Trixi.jl solver [104, 105, 106] to solve Eq. (3.2.1). In Trixi.jl, the entropy-stable Discontinuous Galerkin Spectral Element Method (ES-DGSEM) is used to solve the Euler

and Navier-Stokes equations. For all three problems, we first select two or three geometric and flow parameters and then define an interval for the change. Then, we use random selection methods to create a list of cases for training and testing neural operators. In the following, we explain the specifics of the data generation procedure.

### 3.3.1 Inviscid hypersonic flow around semi-ellipses

We solve the hyperbolic part of Eq. (3.2.1) in a physical domain depicted in Fig. 3.1. According to Fig. 3.1, supersonic inflow is imposed on the blue boundaries while the extrapolation condition is imposed on the red lines. The geometry boundary, defined by the black curve, denotes the slip wall boundary condition. We use two parameters that define the semi-ellipse for generating 36 unique geometries: the major axes ( $a \in [0.5, 3.0]$ ) and the minor axes ( $b \in [0.5, 1.8]$ ) as shown in Fig. 3.2. Latin hypercube sampling is used for defining these 36 samples from the parameter space, see appendix A2.1 for more details. Individual mesh files are generated using an automation script integrated with GMSH. Fig. 3.3 (a)-(c) shows the quadrilateral mesh elements generated for three cases. The reference length size of elements is set at 0.25 for all the points. According to Fig. 3.3, the initial and final meshes for the domain are entirely different due to the adaptive mesh refinement (AMR) strategy.

The flow field is initialized using a free stream condition with Mach number set as  $M = 10$  and  $\rho_0 = 1.4$ ,  $u_0 = -10$ ,  $v_0 = 0$ , and  $p_0 = 1$ . The specific heat ratio is set at  $\gamma = 1.4$ . Shock stabilization is performed using the blending of low and high-order flux functions. The approximate Riemann solver of Lax and Friedrich is employed to patch the advective fluxes across elements. Polynomial order is set at  $P = 3$ , translating into fourth-order spatial accuracy. The ES-DGSEM approach is constructed using the method of Hennemann *et al.*[107]. The mesh refinement is accomplished using the Hennemann indicator[107] applied on  $\rho \times p$  and three levels of refinement with a maximum threshold of 0.1. The AMR is activated every 5 time steps. A positivity preservation limiter is also



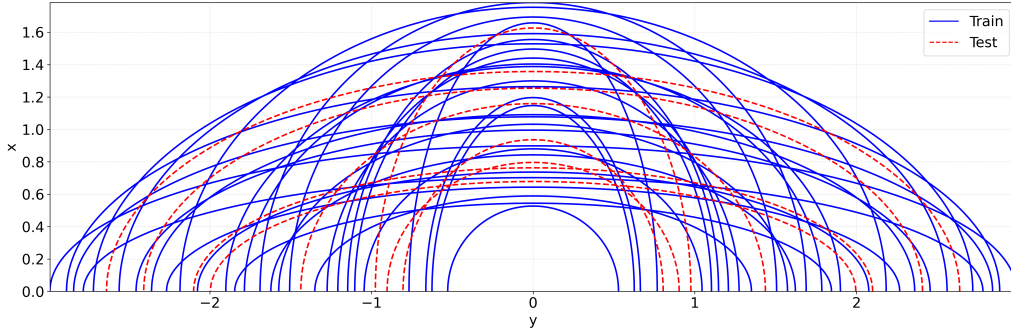
**Figure 3.1:** Schematic of the domain used for solving compressible Euler equations. The blue lines are defined as supersonic inflow boundary condition. The red lines are defined as extrapolation boundary. The black line is a semi-ellipse where the wall-slip boundary conditions are defined.

used for density and pressure. For time stepping, the five-stage fourth-order Carpenter and Kennedy Runge-Kutta approach [108] is used.

Our goal is to map the semi-ellipse’s geometry into the solution of the Euler equations, see an example in Fig. 3.4. We divide the 36 samples into 28 training and 8 test sample sets, see appendix A2.1. Splitting is done using a uniform random distribution sampling. Figure 3.2 depicts the geometries of the semi-ellipse employed for training and test datasets. The solution of the Euler system is computed on an unstructured grid, as shown in Fig. 3.3. Also, the number of grid points varies between samples. We use irregular and regular grid spacing to study the effect of grid point spacing on solution mapping. In the following section, we describe the pre-processing steps applied to the raw data obtained by the Trixi.jl solver to prepare the dataset for operator learning.

### Uniform Cartesian grid

In this study, we evaluate the parameter-conditioned U-Net and FNO neural operators to learn the mapping between geometry parameters and flow fields. Therefore, we used linear interpolation to map the solution from an irregular grid into a structured Cartesian grid that covers the entire domain shown in Fig. 3.1, including the area inside the semi-ellipse. In the  $y$  and  $x$  directions, we employed 256 discretization points and generated



**Figure 3.2:** Geometry of all the semi-ellipses used for training and testing. We have used 28 for training and 8 for testing. The blue lines refer to training samples and the dashed red lines refer to testing samples.

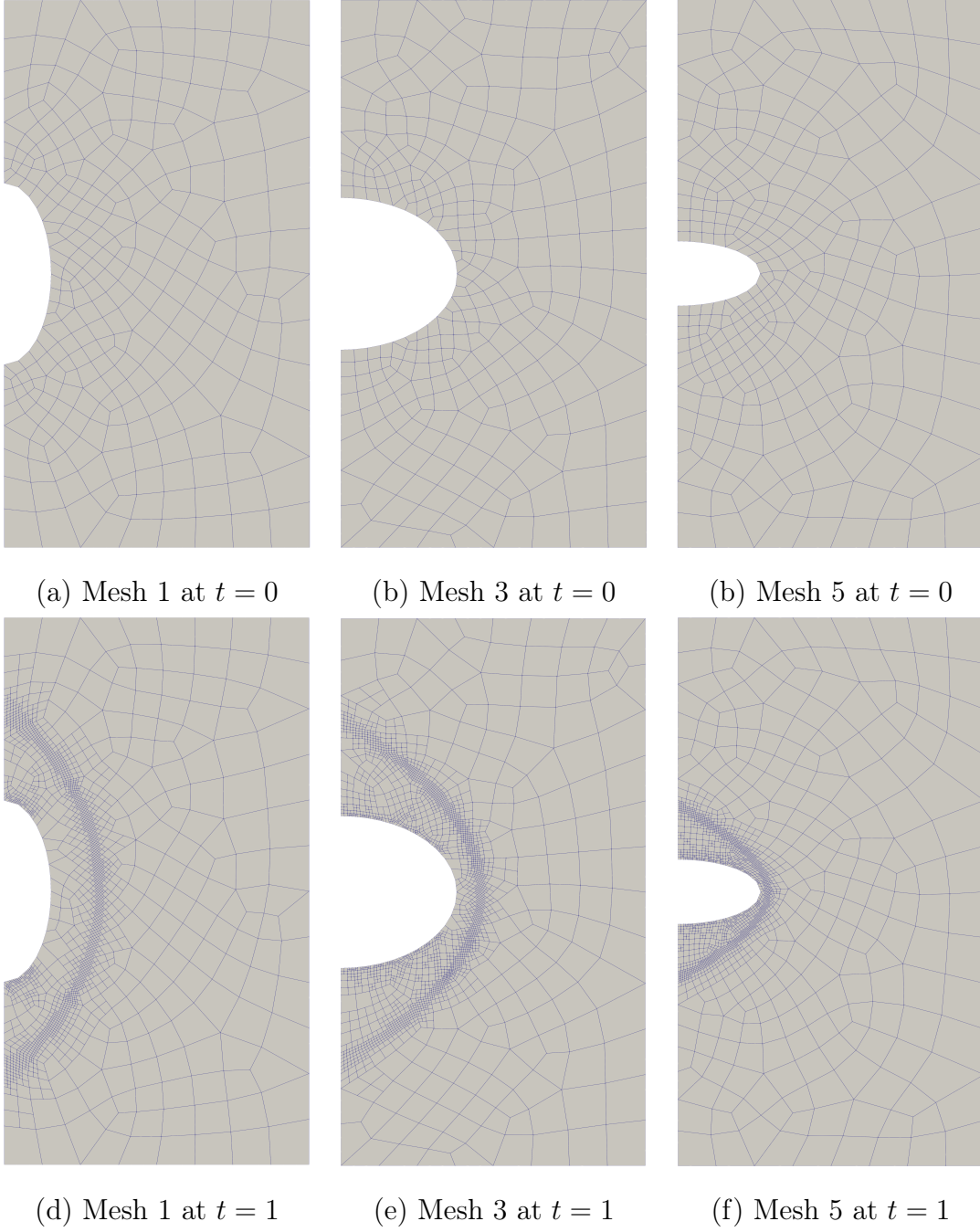
a total of  $256^2$  uniformly spaced grid points. We generate a mask matrix for each case with zero or one entry as outlined in [21]. Zero value is assigned to coordinate points inside the semi-ellipse, and one is assigned to points outside the semi-ellipse and inside the rectangular domain. Finally, all the training and testing dataset samples are projected onto the same Cartesian grid.

### Unstructured irregular Grid

We utilize neural operators like DeepONet and MeshGraphNet, which can directly work with the irregular grids generated by the Trixi.jl solver. These grids adapt dynamically to the solution of the Euler equation, resulting in varying grid point coordinates and number of points for each case. For the DeepONet network, maintaining a consistent input shape for the trunk network is necessary when using a batch size larger than one. To address this, we standardize the number of grid points to match the maximum number of points in the training dataset. If a sample has fewer points, we pad it by repeating its last coordinate to ensure uniformly sized array for operator network training and evaluation.

### 3.3.2 Hypersonic viscous flow around Reentry capsule

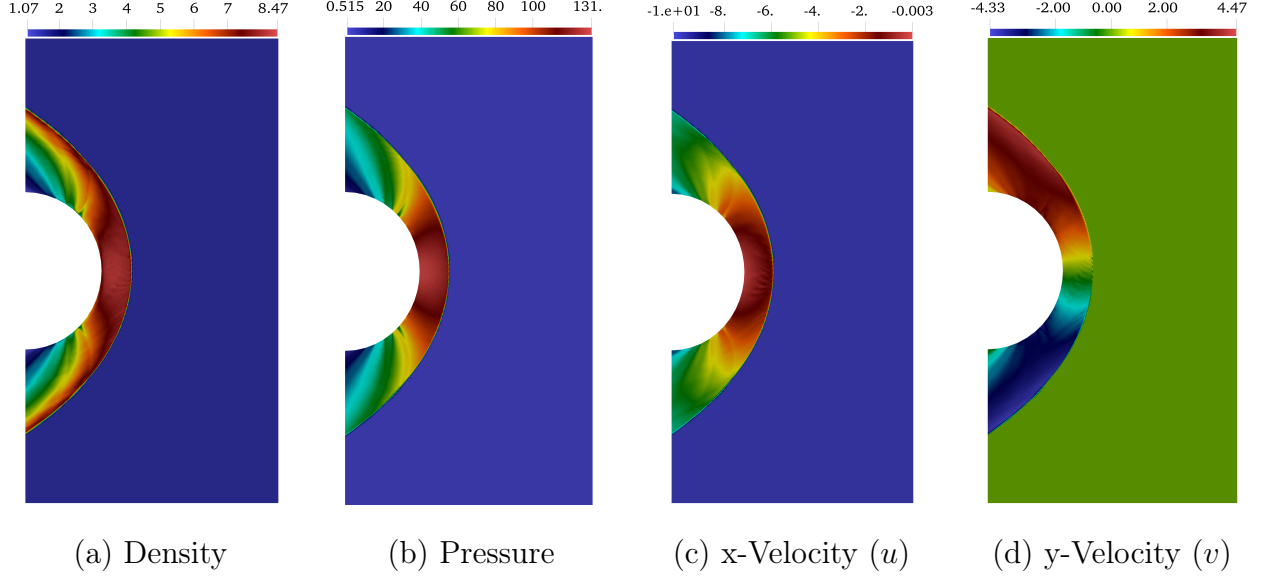
We design a challenging problem for which geometric and flow parameters are considered simultaneously. A  $\beta$ -apex blunt cone geometry [109] is parameterized to generate a flow



**Figure 3.3:** *Initial mesh configuration and adaptive mesh at final time. (a) Mesh case 1 at  $t = 0$ , (b) Mesh case 3 at  $t = 0$ , (c) Mesh case 5 at  $t = 0$ , (d) Mesh case 1 at  $t = 1$ , (e) Mesh case 3 at  $t = 1$ , (f) Mesh case 5 at  $t = 1$ .*

field surrogate. The schematic for the physical domain is shown in Fig. 3.5(a). A supersonic inflow boundary condition is imposed at the blue boundaries, and an extrapolation boundary condition is imposed on the red boundary. The surface of the capsule is selected as an isothermal wall boundary condition with  $T_w = 25 \text{ }^\circ\text{C}$ . According to a magnified

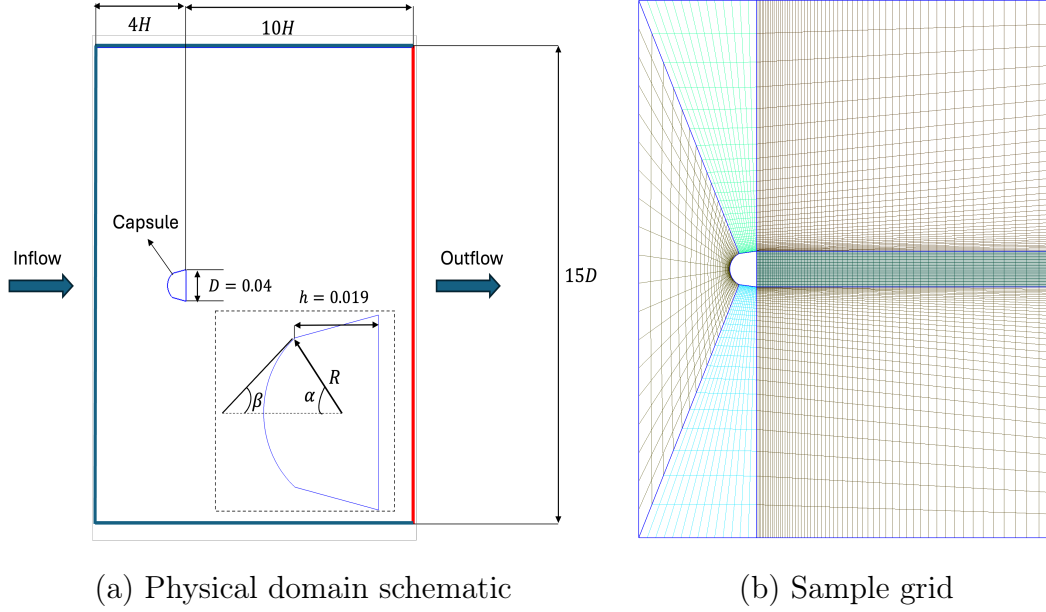




**Figure 3.4:** Solution fields for case 1 at  $t = 1$ . (a) Density contours, (b) Pressure contours, (c) x-Velocity ( $u$ ), (d) y-Velocity ( $v$ ).

view of the capsule in Fig. 3.5(a), the radius ( $R$ ) of the blunt section and the  $\alpha$  angle are designated as geometric parameters that vary such that  $\alpha \in [40^\circ, 60^\circ]$  and  $R \in [13.06, 20]$ . We then added Mach number of the free stream as an extra input parameter defined as  $M \in [5, 8]$ . The free stream conditions are set as  $p_\infty = 850$  Pa,  $\rho_\infty = 0.0141$  kg/m<sup>3</sup>,  $Re_\infty = 4 \times 10^4$  and  $Pr = 0.72$ . The reference length is set as  $L_f = 1$  m.

We employ the Latin hypercube sampling [110] method to randomly draw samples inside the three-dimensional parametric space constructed by  $(R, \alpha, M)$ . We selected 60 samples with various geometries shown in Fig. 3.6 and free-stream Mach numbers. For each of the 60 samples, we automatically generated the geometry of the physical domain using GMSH and generated a structured mesh with only quadrilateral elements. We discretize the physical domain using 8,501 elements in total, see Fig. 3.5(b). We then employ the Trixi.jl solver to solve Eq. 3.2.1 for twelve flow-through times to allow the flow to reach steady-state conditions. Each flow-through time is defined as the non-dimensional time that the free-stream flow starts from the inflow boundary and exits from the outflow boundary, and can be computed as  $14H/M$ . At the end of the simulations, we select the last snapshot of the time-dependent solution as the steady-state flow field that the

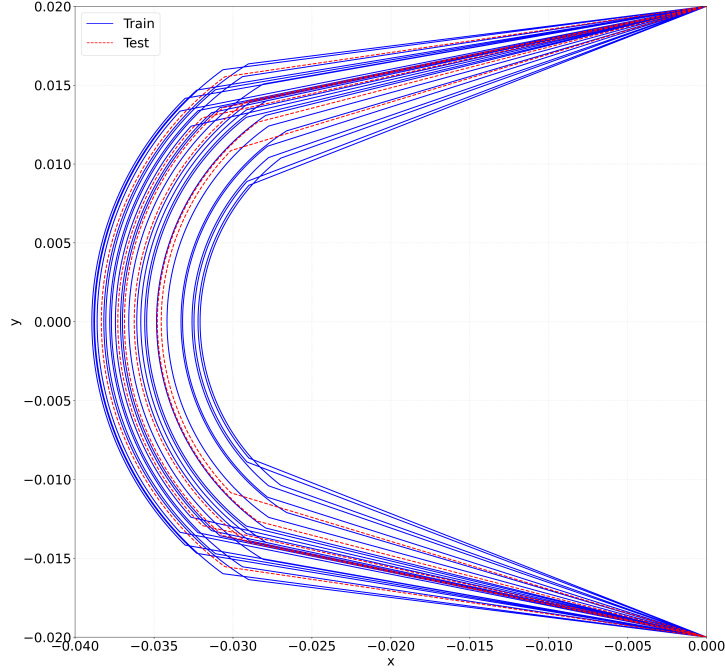


**Figure 3.5:** (a) Schematic of the domain used for solving compressible Navier-Stokes equations. A capsule geometry is placed inside a rectangle. Supersonic inflow boundary conditions are imposed at the blue boundaries. Supersonic outflow is also imposed at the red boundary. Isothermal wall boundary condition is imposed on the capsule surfaces. A zoomed-in view of the capsule is shown inside the dashed outline box. The geometry of the capsule is parameterized using  $\alpha \in [40, 60]$  and  $R \in [13.06, 20]$  and the flow condition is parametrized using Mach number as  $M \in [5, 8]$ . All the dimensions are in meters and  $H = 0.03177$ . (b) Domain discretization using quadrilateral elements

machine learning model will learn. After completing the simulations, we create a geometry and Mach-dependent flow field dataset. Since Trixi.jl employs the DGSEM scheme, it uses nodal solution points on the sides of quadrilateral elements. Therefore, duplicate solution points exist on the elements' internal edges. We eliminate all duplicate points before training the machine learning model. From 60 samples, we randomly select 48 samples for training and 12 for testing datasets, see appendix A2.1 for dataset details.

### 3.3.3 Viscous flow inside convergent-divergent nozzle

The last example includes learning the viscous flow field for an internal flow problem. The physical domain consists of a converging-diverging nozzle attached to a duct as shown in Fig. 3.7. The compressible flow enters the nozzle at Mach 0.6 and a Reynolds number of 1000. Although the inflow Mach number is set at 0.6, the flow transforms to



**Figure 3.6:** Geometry of all capsules used for operator network learning. We use 48 samples for training and 12 for testing, see appendix A2.1. The blue lines refer to training samples and the dashed red lines refer to testing samples.

a supersonic regime in the diverging part of the nozzle. Inflow conditions are defined as  $p_\infty = 89214.769$  Pa and  $\rho_\infty = 1.074\text{kg/m}^3$  and gas constant  $R = 287.0$  J/Kg/K. The  $Pr = 0.72$  is selected, and the isothermal wall temperature is set at  $T_w = 2T_\infty$ . At the inflow boundary, we impose subsonic inflow boundary condition [111], and at the outflow boundary, the subsonic outflow [111]. The physical domain is discretized using quadrilateral elements with a polynomial order of 3 for the DGSEM solver. We initialize the entire domain using the inflow conditions and simulate for 20 flow-through times to reach a steady state solution.

Our objective is to generate a surrogate model that takes three geometric parameters and predicts flow field variables such as temperature, density, pressure, and velocity. The geometric parameters consist of inlet height  $2h_i$ , outlet height  $2h_o$ , and the x-coordinate of the location of the throat area of the converging-diverging nozzle ( $10x_t$ ). The three parameters vary such that  $h_i \in [3, 4]$ ,  $h_o \in [1.5, 2.5]$ , and  $x_t \in [0.45, 0.65]$ . By changing the geometric parameters, the overall shape of the nozzle is constructed using a fifth-

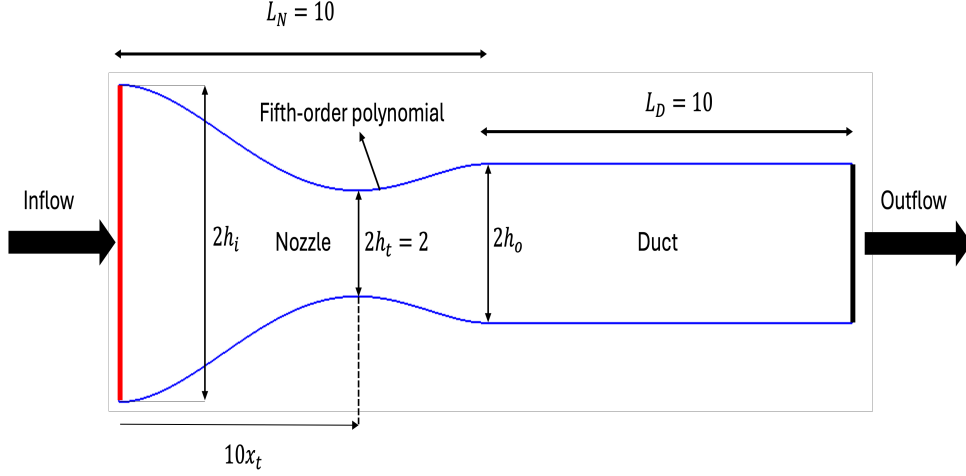
order polynomial. The top wall of the nozzle is constructed by a fifth-order polynomial constrained by the following geometric conditions:

$$\begin{aligned}
y &= a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0, \\
y(x=0) &= \frac{h_i}{2}, \quad y(x=L_N) = \frac{h_o}{2}, \quad y(x=x_t) = 1 \\
\frac{dy}{dx}\Big|_{x=0} &= 0, \quad \frac{dy}{dx}\Big|_{x=L_N} = 0, \quad \frac{dy}{dx}\Big|_{x=x_t} = 0.
\end{aligned} \tag{3.3.1}$$

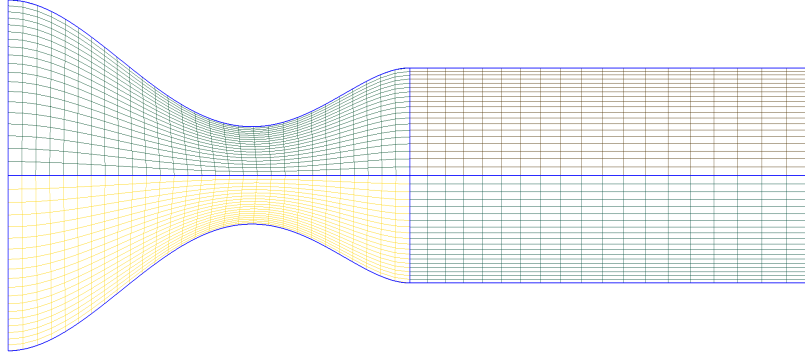
The bottom wall is the mirror image of the top wall. We draw 60 samples using Latin hypercube sampling from the three-dimensional parametric space and generate the geometry for all the cases using GMSH. We then discretize the domain using quadrilateral elements that are clustered close to the isothermal walls using 1824 elements. From the 60 generated cases, we randomly select 70% (i.e., 42) trajectories for training and 18 for testing datasets, see Fig. 3.8 and appendix A2.1 for dataset details. Next, we compute the steady-state solution using Trixi.jl and extract the solution field including  $p$ ,  $\rho$ ,  $u$ ,  $v$ , and  $T$  from the output files. We remove all the duplicate points in the domain since the spectral element method uses nodes on the common edges of the computational elements. Ultimately, we accumulate the solution, coordinates, and geometric parameters into three separate tensors to feed into the training loop.

### 3.4 Methodology

Here, we describe the architecture of Fusion-DeepONet model and how to improve its ability to predict not only the solution field but also the derivative of the solution field, specifically close to the boundary of the physical domain. Our quantities of interest in hypersonic flows are usually shear stress, pressure, and heat flux acting on the surfaces of the flying object. By predicting the quantities of interest on the solid surfaces, we can optimize the geometry to reduce drag force, increase lift force, and minimize heat flux going into the body. Therefore, we aim to develop a method to accurately predict the



(a) Physical domain schematic



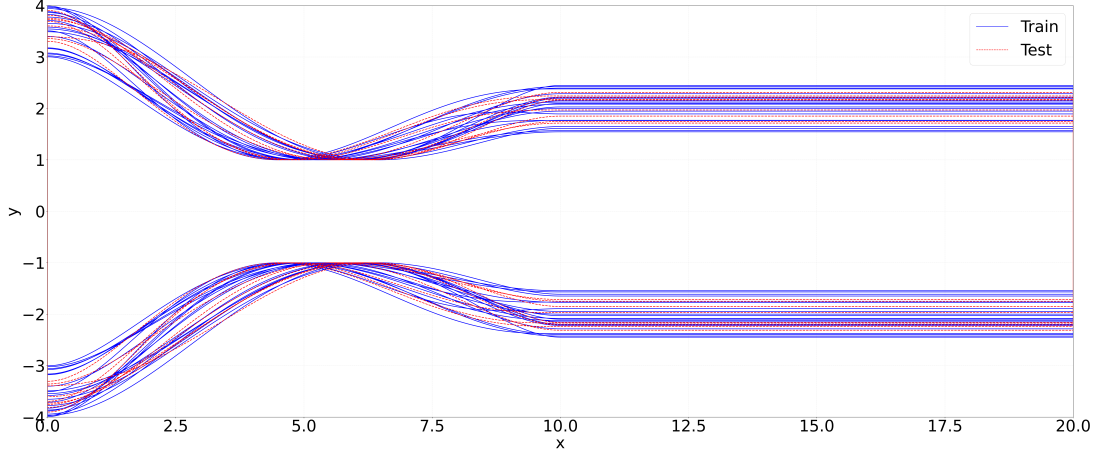
(b) Sample grid

**Figure 3.7:** (a) Schematic of the domain used for solving compressible Navier-Stokes equations. The domain consists of a converging-diverging nozzle and a duct. The length of the nozzle is 10 and the length of the duct is 10. The throat diameter is set at 2. The inflow Mach number is 0.6 and the Reynolds number is 1000 based on a reference length scale of one. The wall boundary condition is isothermal and  $T_w = 2T_\infty$ . The walls of the nozzle are represented using a fifth-order polynomial by imposing sixth geometric constraints. For this problem, we use nozzle inlet diameter  $h_i$ , nozzle outlet diameter  $h_o$  and the location of the throat  $x_t$  as the geometric parameters. (b) A sample discretization of the physical domain.

flow field and its derivatives on the physical boundary of the varying physical domain.

### 3.4.1 Fusion-DeepONet

We present Fusion-DeepONet, a multi-scale conditioned neural field. Conditional neural fields were first conceptualized by Wang *et al.* [112], who introduced the idea of global and local conditioning in neural fields. DeepONet, in this context, can be characterized as



**Figure 3.8:** Geometry of all the converging diverging nozzles used for operator network learning. We use 42 samples for training and 18 for testing, see appendix A2.1 for dataset details. The blue lines refer to training samples and the dashed red lines refer to testing samples.

a globally conditioned neural field. In DeepONet, the trunk network serves as the neural field, with its final linear layer being conditioned by the output of the branch network. Conditioning the neural field at its final layer for all coordinate points defines global conditioning. Additionally, Seidman *et al.* [113] proposed the NoMaD architecture, which conditions the neural field locally (for each coordinate point) by concatenating the input function encoding with the input encoding of the neural field. In Fusion-DeepONet, we introduce a new type of conditioning that conditions each trunk network hidden layer using the latent encoding of branch hidden layers while keeping the global conditioning of the trunk network output. Below, we explain the architecture of the Fusion-DeepONet.

Fusion-DeepONet, like the Vanilla-DeepONet, is built from two sub-networks: a branch network that processes the input function and a trunk network that processes the coordinates. We begin by defining the architectures of each sub-network. For  $L \in \mathbb{N}$  and the vector  $\vec{n}^b = (n_0^b, n_1^b, \dots, n_L^b * n_v) \in \mathbb{N}^{L+1}$  with  $n_v$  denoting number of output variables and  $n_l^b = n$  for  $l = 1, \dots, L-1$  denoting the width of layer  $l^{\text{th}}$ , an  $L-1$ -layer feed-forward branch network is given recursively by

$$\begin{aligned} \mathbf{Z}^0 &= \mathbf{x}^b \\ \mathbf{Z}^l &= \phi(\mathbf{W}^{b,l} \mathbf{Z}^{l-1} + \mathbf{b}^{b,l}, \mathbf{a}^{b,l}), \quad l = 1, \dots, L-1, \end{aligned} \tag{3.4.1}$$

where  $\phi(\cdot, \mathbf{a}^{b,l})$  is an adaptive activation with learnable parameters  $\mathbf{a}^{b,l} \in \mathbb{R}^{N_R}$  with  $N_R$  denoting number of Rowdy activation function learnable coefficients. In Eq. (3.4.1),  $\mathbf{x}^b \in \mathbb{R}^{N \times N_p}$  denotes the vector of branch inputs, i.e., geometric or flow parameters, with  $N$  training samples and  $N_p = n_0$  number of input parameters. The weight matrix is defined as  $\mathbf{W}^{b,l} \in \mathbb{R}^{n_{l-1}^b \times n_l^b}$  for  $l = 1, \dots, L-1$ . The bias matrix is also defined as  $\mathbf{b}^{b,l} \in \mathbb{R}^{1 \times n_l^b}$ . The matrix  $\mathbf{Z}^l \in \mathbb{R}^{N \times n_l^b}$  denotes the output of branch network layers for  $l = 1, \dots, L$ . Similarly, for  $\vec{n}^t = (n_0^t, n_1^t, \dots, n_L^t) \in \mathbb{N}^{L+1}$  with  $n_l^t = n$  for  $l = 1, \dots, L-1$ , the trunk network is constructed as

$$\begin{aligned} \mathbf{Y}^0 &= \mathbf{x}^t, \\ \mathbf{Y}^l &= \phi(\mathbf{W}^{t,l} \mathbf{Y}^{l-1} + \mathbf{b}^{t,l}, \mathbf{a}^{t,l}), \quad l = 1, \dots, L-1, \end{aligned} \quad (3.4.2)$$

where  $\phi(\cdot, \mathbf{a}^{t,l})$  is an adaptive activation with learnable parameters  $\mathbf{a}^{t,l} \in \mathbb{R}^{N_R}$  with  $N_R$  denoting number of Rowdy activation function learnable coefficients. In Eq. (3.4.2),  $\mathbf{x}^t \in \mathbb{R}^{N \times N_{pts} \times N_c}$  denotes the vector of trunk inputs, i.e., coordinate points for training samples, with  $N$  training samples,  $N_{pts}$  number of solution points for each sample, and  $N_c = n_0^t$  number of coordinate directions. The weight matrix is defined as  $\mathbf{W}^{t,l} \in \mathbb{R}^{n_{l-1}^t \times n_l^t}$  for  $l = 1, \dots, L-1$ . The bias matrix is also defined as  $\mathbf{b}^{t,l} \in \mathbb{R}^{1 \times n_l^t}$ . The tensor  $\mathbf{Y}^l \in \mathbb{R}^{N \times N_{pts} \times n_l^t}$  denotes the output of trunk network's layers for  $l = 1, \dots, L-1$ .

Next, we introduce connection coefficients that condition the trunk net layers. Define

$$\begin{aligned} \mathbf{S}^0 &= \mathbf{Z}^1, \\ \mathbf{S}^m &= \mathbf{Z}^{m+1} + \mathbf{S}^{m-1}, \quad m = 1, \dots, L-2, \end{aligned} \quad (3.4.3)$$

where  $\mathbf{S}^m \in \mathbb{R}^{N \times n_{m+1}^b}$ . We want to condition the output of trunk layers  $\mathbf{Y}^l \in \mathbb{R}^{N \times N_{pts} \times n_l^t}$  using  $\mathbf{S}^m \in \mathbb{R}^{N \times n_{m+1}^b}$ ; however, the tensor dimensions does not match at the first glance and we previously assumed that number of neurons follows  $n_{m+1}^b = n_l^t = n$ . To be able to perform element-wise multiplication, we add a single axis to  $\mathbf{S}^m \in \mathbb{R}^{N \times n}$  and construct  $\tilde{\mathbf{S}}^m \in \mathbb{R}^{N \times 1 \times n}$  to account for the extra axis of trunk output that accounts for coordinate

points. Then conditioning proceeds by element-wise multiplication,

$$\hat{\mathbf{Y}}^l = \mathbf{Y}^l \odot \tilde{\mathbf{S}}^{l-1}. \quad l = 1, \dots, L-2. \quad (3.4.4)$$

This element-wise multiplication conditions the output of the trunk layer over the first axis that accounts for a number of samples ( $N$ ) and the last axis that accounts for the number of neurons using unique coefficients while using common coefficients for all the coordinate points. In Eq. (3.4.4),  $\hat{\mathbf{Y}}^l \in \mathbb{R}^{N \times N_{pts} \times n_l^t}$  denotes the conditioned trunk layer output. Finally, following the DeepONet convention, the last layers are linear:

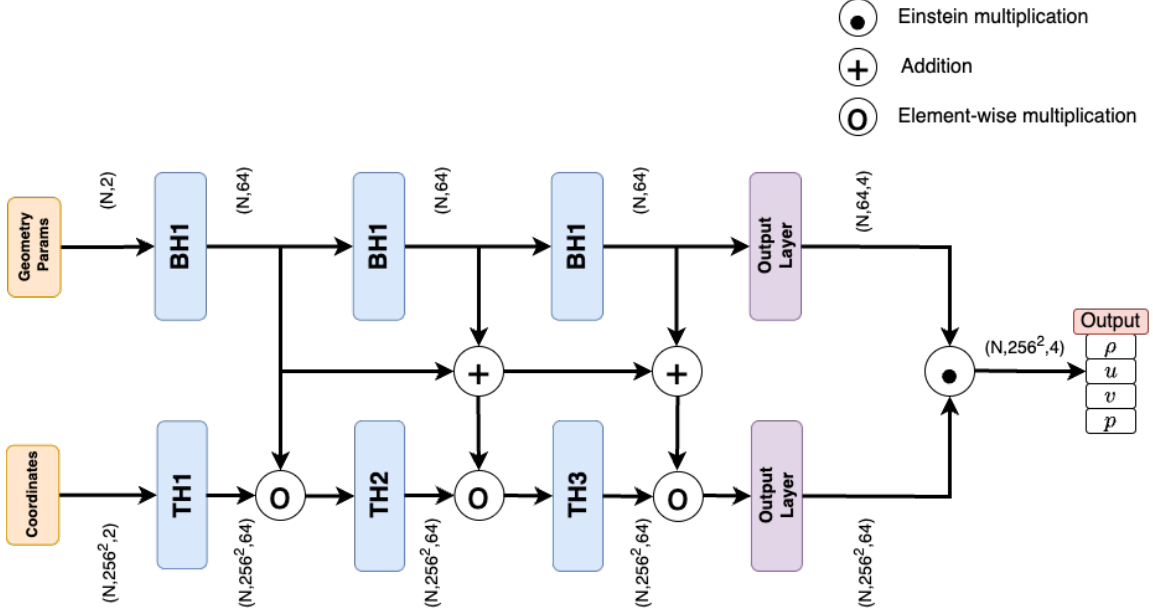
$$\begin{aligned} \mathbf{Z}^L &= \mathbf{W}^{b,L} \mathbf{Z}^{L-1} + \mathbf{b}^{b,L}, \\ \hat{\mathbf{Y}}^L &= \mathbf{W}^{t,L} \hat{\mathbf{Y}}^{L-1} + \mathbf{b}^{t,L}. \end{aligned} \quad (3.4.5)$$

The Fusion-DeepONet output is then the dot product of these two final vectors:

$$\mathcal{G}(\mathbf{x}^b, \mathbf{x}^t) = \mathbf{Z}^L \cdot \hat{\mathbf{Y}}^L.$$

For Fusion-DeepONet, every hidden layer output of the neural field is conditioned using the branch hidden layer output while retaining the global conditioning of the trunk network. In Fusion-DeepONet, as shown in Fig. 3.9, the output of the first hidden layer of the branch is used to condition the output of the first, second, and third hidden layers of the trunk network. As mentioned in the study by Peyvan *et al.* [8], the first layers of the fully connected neural nets learn the low-mode features of the target data. Therefore, the low-mode features encoded by the branch layer modulate the low-to-highest-mode functions of the basis functions of the target solution; hence, the conditioning of the neural field occurs on multiple scales. Cross-scale conditioning of the neural field significantly improves the accuracy of the geometry-dependent discontinuous flow field. We later show that the fusion of branch layers into a trunk significantly reduces the over-fitting due to the limited training data.





**Figure 3.9:** *Fusion-DeepONet* schematic. All the tensor dimensions are shown.  $BH_I$  refers to the branch  $I^{th}$  hidden layer and  $TH_I$  denotes trunk net  $I^{th}$  hidden layer.  $256^2$  denotes the number of grid points used in the uniform grid. For the irregular grid, the number of points is the maximum point count in the training dataset.  $N$  denotes number of training samples.

According to Fig. 3.9, we assume that the number of hidden layers in the branch and trunk networks are the same and that the width of the hidden layers of the trunk and branch networks are the same, except for the last linear layer of the branch network. The width of the last linear layer of the branch network is selected to account for multiple target fields such as density, x-velocity ( $u$ ), y-velocity ( $v$ ), pressure ( $p$ ), or temperature ( $T$ ). The Fusion-DeepONet consists of extra connections and pathways among the neurons of the neural operators. Therefore, we expect that the forward and backward passes of the Fusion-DeepONet will require extra computational cost compared to the Vanilla-DeepONet for the same number of trainable parameters. We emphasize that the modulation of the hidden layers of the trunk network comes from the branch layers upstream of the current hidden layer of the trunk.

### 3.4.2 Derivative enhanced loss function

To accurately predict the shear stress and heat flux on the solid surfaces in the physical domain, we train the neural operator in a way that can simultaneously learn the solution field, such as temperature, and also the spatial gradients such as  $\partial T/\partial x$  and  $\partial T/\partial y$ . Training the neural operator to learn, for instance, temperature, does not guarantee accurate prediction of the temperature derivative with respect to spatial coordinates. Therefore, we modified the loss function so the model would be aware of the spatial derivative during the training updates. Qiu *et al.*[103] introduced a derivative-enhanced DeepONet (DE-DeepONet), which integrates derivative information (Gâteaux derivatives) into the MSE loss function to improve the accuracy of predicting the directional derivative of the neural operators with respect to the branch input parameters. In this study, we wish to improve the prediction accuracy of the machine learning model of the derivative with respect to the spatial coordinates. Therefore, we introduce a derivative-enhanced loss (DEL) function that informs the neural operator about the derivative of the solution field.

Let  $\hat{\mathcal{G}}(x, y, m; \theta)$  denotes the output of the neural operator with  $\theta$  trainable parameters of the network and  $m$  the input geometric parameter; we minimize the following loss function

$$L(\theta) = \|\mathcal{G}(m) - \hat{\mathcal{G}}(m; \theta)\|_{L^2(\Omega)}^2 + \lambda_1 \|d\mathcal{G}(m) - \hat{d}\hat{\mathcal{G}}(m; \theta)\|_{L^2(\Omega)}^2, \quad (3.4.6)$$

where the  $L^2$  norm of the function  $f$  is defined as  $\|f\|_{L^2(\Omega)}^2 = \int_{\Omega} |f(\mathbf{x})|^2 d\mathbf{x}$ . In Eq. (3.4.6),  $d$  and  $\hat{d}$  denote derivative operators that act on the ground truth and the neural operator prediction, respectively. In this study, we investigate the best approach to generate a derivative operator to use in the derivative loss to enhance the accuracy of Fusion-DeepONet in predicting the derivation of the flow field. We evaluate several approaches that best represent the derivative operation with the least computational costs.

First, we use the same operator for  $d$  and  $\hat{d}$ , and represent the derivative using the following formulation. Let  $(x_i, y_i)$  and  $(x_{i+1}, y_{i+1})$  be two consecutive coordinate points in

the tensor of coordinates, and  $\phi_i$  and  $\phi_{i+1}$  be the values of the predicted solution on these two points. We construct a derivative operator such that

$$d = \hat{d} = \frac{\phi_i - \phi_{i+1}}{\sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2}}. \quad (3.4.7)$$

Therefore, we apply this operator to both the predicted value by the neural operator and the ground truth. This derivative operator is straightforward; however, it provides useful information to force the neural operator to learn derivative information. This operator is not computationally expensive and can be vectorized. It is clear that this operator does not represent the discrete form of  $\nabla\phi$ ; however, it provides necessary information to improve the neural operator's predictability for the field's derivatives.

Second, since we train neural operators using discrete points from a point cloud structure, we define a local least-squares approximation of the gradient at each point. Then, we project that gradient onto a given direction  $\mathbf{v}$  to compute the directional derivative. We calculate the directional spatial derivative of the output using the finite differencing method. For each point  $i$ , we first find  $k$  closest points. At each point  $i$ , we approximate the true scalar field  $u(x, y)$  by a local linear model

$$u(\mathbf{x}_i + \Delta\mathbf{x}) \approx u(\mathbf{x}_i) + \nabla u(\mathbf{x}_i) \cdot \Delta\mathbf{x}. \quad (3.4.8)$$

Now, we estimate  $\nabla u(\mathbf{x}_i)$  from the  $k$  neighbors around  $\mathbf{x}_i$ . Let  $\mathbf{A}_i$  be a  $k \times 2$  matrix for each central point  $i$ . The rows of matrix  $\mathbf{A}_i$  are the displacement vectors  $\Delta\mathbf{x}_{i \rightarrow i_j} = \mathbf{x}_{i_j} - \mathbf{x}_i$  and  $\mathbf{b}_i$  be a length- $k$  vector of the corresponding changes in the scalar field  $\Delta u_{i \rightarrow i_j} = u(\mathbf{x}_{i_j}) - u(\mathbf{x}_i)$ . We solve a least squares problem

$$\min_{\mathbf{g}} \|\mathbf{A}_i \mathbf{g} - \mathbf{b}_i\|_2^2 \quad (3.4.9)$$

for the gradient vector  $\mathbf{g}$ . The matrix form for Eq. (3.4.9) can be expressed as

$$(\mathbf{A}_i^\top \mathbf{A}_i) \mathbf{g} = \mathbf{A}_i^\top \mathbf{b}_i, \quad (3.4.10)$$

thus

$$\nabla u(\mathbf{x}_i) \approx \mathbf{g}_i = (\mathbf{A}_i^\top \mathbf{A}_i)^{-1} \mathbf{A}_i^\top \mathbf{b}_i. \quad (3.4.11)$$

Finally, the directional derivatives can be computed by

$$D_v u(\mathbf{x}_i) = \nabla u(\mathbf{x}_i) \cdot v. \quad (3.4.12)$$

In summary, this is a scattered-data finite-difference scheme: we fit a local linear model around each point using its nearest neighbors, then read off the directional derivative from that model. To reduce the computational cost, we precompute the nearest points' index and the ground truth's directional gradient. For this approach, we apply this least squares derivative operator to discretely compute the directional derivative of the neural operator prediction and ground truth. In the results section, we investigate the accuracy and computational cost of the two approaches to enhance the derivative predictability of the neural operator.

## 3.5 Results

This section presents the results of three geometry-dependent surrogate modeling of high-speed flows using Fusion-DeepONet. We start with two hypersonic problems: inviscid hypersonic flow around semi-elliptic blunt geometries and hypersonic viscous flow over a reentry capsule. We then finish the section by learning an internal supersonic flow in a converging-diverging nozzle connected to an isolator duct. The flow field of all three problems involves strong discontinuities in the solution, where we have shock waves, which

are challenging for operator networks to learn. Additionally, using scarce training data in training severely impacts the generalization error of such networks. According to Figs. 3.3, 3.6, and 3.8, the geometry in all three problems changes significantly from case to case, and along with it, the grid points change. Learning such complex input-output mapping is essential for real-world geometry optimization problems, such as designing complex 3D geometry of aerial vehicles traveling at hypersonic speeds.

### 3.5.1 Inviscid hypersonic flow around semi-ellipses

For this problem, we randomly selected 28 cases for training and 8 cases for testing. We used the same training and testing datasets for all neural operators. We train six different neural operators, including Vanilla-DeepONet, POD-DeepONet, and Fusion-DeepONet with Rowdy adaptive activation functions, parameter-conditioned U-Net, FNO, and MeshGraphNet with individual architecture setups discussed in Appendix A2.4. We used the mean squared error of the solution field as the loss function for training all operator models.

We have three versions of the vanilla, POD-DeepONet, and Fusion-DeepONets that are learning the hypersonic field over uniform and irregular grids. The architecture details of all variants of DeepONet are shown in Table 3.1. For Fusion and Vanilla-DeepONet, we use three hidden layers with 64 neurons in the branch network and three with 64 neurons in the trunk network. The adaptive Rowdy activation function [8] is employed for both trunk and branch networks. The base function for the rowdy activation function is  $\tanh$ . Adam optimizer and a variable learning rate are used to train the DeepONet framework for 50,000 epochs. We vary the learning rate using the exponential decay function with the step size, decay step, and decay rate set as 0.001, 2000, and 0.91 respectively. For the POD-DeepONet, we should note that there is no trainable trunk network; we only use the basis functions computed by the singular value decomposition (SVD) of the 28 training samples. For the last layer of the branch network, we select the number of neurons to be

$64 \times 4$ , where 4 corresponds to the four variables predicted by the neural operator.

For the uniform grid, we input  $256^2$  coordinates with shape  $(28, 65536, 2)$  into the trunk net and a matrix that contains the values of  $a$  and  $b$ , which are the geometric parameters with the shape of  $(28, 2)$  into the branch network. In the case of a uniform Cartesian grid, we multiply the mask matrix into the difference of predictions and the ground truth to zero out the values inside the semi-ellipse and diminish their effect on the update of the neural network parameters. The Vanilla and Fusion-DeepONets then predict all four variables, including density, x-direction velocity, y-direction velocity, and pressure, at the same time using a single trunk network. However, for the POD-DeepONet, we employ four trunk networks, assigning each to the density prediction, x-direction velocity, y-direction velocity, and pressure. For the irregular grid, the input of the trunk net is a tensor with the shape of  $(28, npts, 2)$  where the  $npts$  refers to the maximum number of grid points across the training samples to learn the flow over an irregular grid. The prediction output will then take a shape as  $(28, npts, 4)$ , where the last axis refers to  $\rho$ ,  $u$ ,  $v$ , and  $p$ , respectively. The total number of parameters used for Vanilla and Fusion-DeepONet is 37,854, including the weights and biases, and the learnable parameters of the Rowdy activation function.

**Table 3.1:** *Architecture details for all variants of DeepONets*

| Variants | Trunk layers        | Branch Layers         | Activation |
|----------|---------------------|-----------------------|------------|
| Vanilla  | [2, 64, 64, 64, 64] | [2, 64, 64, 64, 64*4] | Rowdy      |
| POD      | [28*4]              | [2, 64, 64, 64, 28*4] | Rowdy      |
| Fusion   | [2, 64, 64, 64, 64] | [2, 64, 64, 64, 64*4] | Rowdy      |

Table 3.2 shows the prediction error for the testing dataset and the computational time of training. All the training is performed using a single NVIDIA 3090 GeForce RTX GPU. The hypersonic flow is predicted on the uniform and irregular grid. Therefore, we compare various neural operators trained using both types of grids. Focusing on the uniform grid, Fusion-DeepONet shows the best accuracy in predicting all four variables compared to

**Table 3.2:** *Relative  $L^2$  norm of the primitive variables errors for different frameworks showing the accuracy achieved. GS refers to Grid Spacing, #P refers to number of trainable parameters.*

| Framework        | GS        | #P           | % $L^2(\rho)$ | % $L^2(u)$  | % $L^2(v)$  | % $L^2(p)$  | % $L^2$     | t(sec)  |
|------------------|-----------|--------------|---------------|-------------|-------------|-------------|-------------|---------|
| Fusion-DeepONet  | Uniform   | 37,854       | <b>3.41</b>   | <b>0.68</b> | 5.91        | 4.80        | <b>3.70</b> | 1471.05 |
| Vanilla-DeepONet | Uniform   | 37,854       | 33.2          | 8.43        | 52.4        | 51.57       | 36.41       | 78.6    |
| POD-DeepONet     | Uniform   | 15,807       | 25.29         | 5.46        | 39.21       | 38.76       | 27.18       | 177.4   |
| U-Net            | Uniform   | 7,056,116    | 3.68          | 2.39        | 9.36        | 2.74        | 4.54        | 3164.9  |
| FNO              | Uniform   | 1,71,556,804 | 10.4          | 8.41        | 1.97        | 12.08       | 8.21        | 1650.3  |
| Fusion-DeepONet  | Irregular | 37,854       | <b>6.18</b>   | <b>2.79</b> | <b>9.29</b> | <b>7.29</b> | <b>6.39</b> | 536.2   |
| Vanilla-DeepONet | Irregular | 37,854       | 39.89         | 16.77       | 52.66       | 47.48       | 39.20       | 369.78  |
| MeshGraphNet     | Irregular | 277,061      | 47.33         | 30.22       | 58.81       | 62.40       | 49.46       | 1178.3  |

Vanilla-DeepONet, parameter-conditioned U-Net, and FNO neural operators. Fusion-DeepONets and U-Nets provide comparable results, while FNO falls short in matching the prediction accuracy compared to these two frameworks on a uniform grid. The Vanilla-DeepONet performs the worst. In terms of computational cost, Vanilla-DeepONet has the lowest computational cost compared to other frameworks. We also performed hyperparameter tuning for the Vanilla- and POD-DeepONet to show that even with tuning these two operators cannot learn the semi-ellipse problem. A detailed discussion of hyperparameter tuning and application of regularizers is presented in Appendix A2.3. The Fusion-DeepONet employs the least number of parameters compared to U-Net and FNO while providing comparable accuracy to U-Net. The computational time for training Fusion-DeepONet over 50,000 epochs is higher compared to Vanilla-DeepONet. This is due to two main reasons. First, the forward and backward passes in Fusion-DeepONet involve addition and multiplication operations defined in Eqs. (3.4.3) and (3.4.4). Second, the input to the trunk network in Fusion-DeepONet has a shape of  $(28, 65536, 2)$ , whereas in Vanilla-DeepONet it is  $(65536, 2)$ . In Vanilla-DeepONet, the trunk network is not conditioned on the output of the branch network, allowing the use of the same set of spatial coordinates across all 28 training samples. As a result, Fusion-DeepONet incurs a higher computational cost. However, in scenarios where each sample has a unique

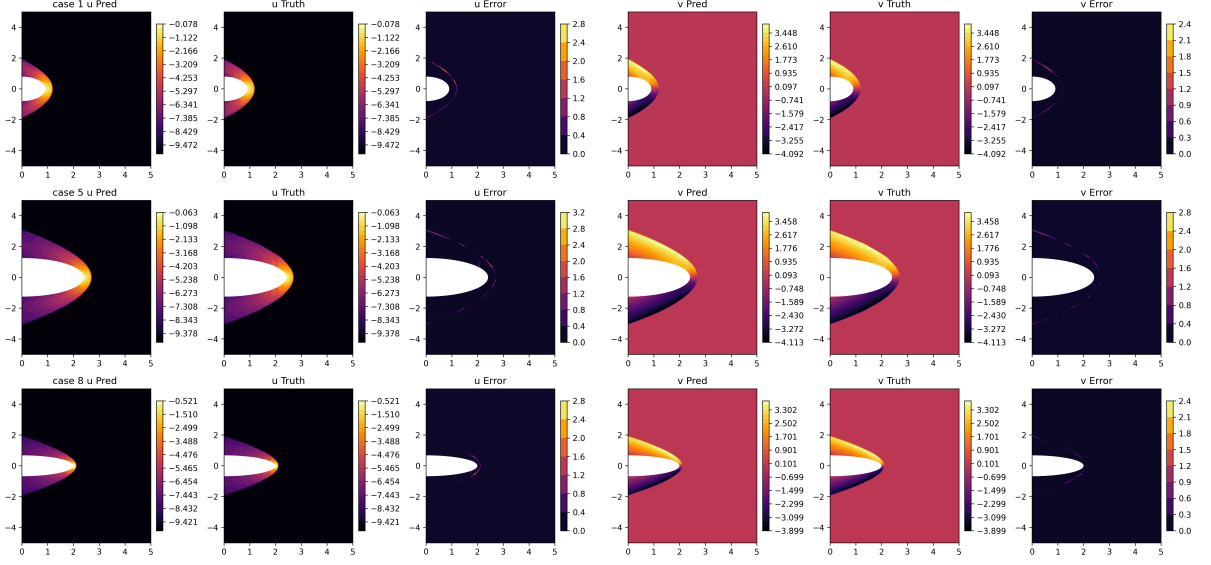
coordinate grid (e.g., irregular grids), Fusion-DeepONet offers higher accuracy at the cost of about a 30% increase in computational overhead compared to Vanilla-DeepONet. In conclusion, Fusion-DeepONet predicts accurately and employs minimum trainable parameters compared to U-Net and FNO.

Learning flow fields on irregular grids presents unique challenges, as only a limited number of operator networks can handle unstructured grid coordinates without requiring additional pre-processing. Among these, Vanilla-DeepONet and the more recent MeshGraphNet are capable of operating on arbitrary mesh grids for output functions. Fusion-DeepONet, being a variant of Vanilla-DeepONet, retains its core capabilities and is therefore included in this comparison. For a fair evaluation, we use the same number of parameters for both the vanilla and Fusion-DeepONet models on irregular grids as in their uniform grid counterparts. As shown in Table 3.2, Fusion-DeepONet is the only model that achieves accurate predictions of output flow fields with limited training data. It consistently outperforms both MeshGraphNet and Vanilla-DeepONet across all predicted variables while also requiring lower computational cost for training. Additionally, we assess Fusion-DeepONet’s performance on a high-pressure ratio Sod shock tube problem, as introduced in [8]. The results are detailed in Appendix A2.2.

### **Training on uniform grid**

In this section, we dive deeper into the predicted results using a uniform grid and try to explain the reason behind the exceptional performance of the Fusion-DeepONet framework. According to Fig. 3.10, the Fusion-DeepONet framework can accurately predict the hypersonic flow field for a significant geometrical variation between test cases. Figure 3.11 compares prediction accuracy among Fusion-DeepONet, U-Net, and FNO for an unseen sample by the neural operators. The point-wise error plot shows that the location of the bow shock is accurately predicted; otherwise, the thickness of the high error region at the bow shock would be wider than what is shown in the figure. Therefore, the nature of the error indicates overshoots or undershoots in the case of Fusion-DeepONet

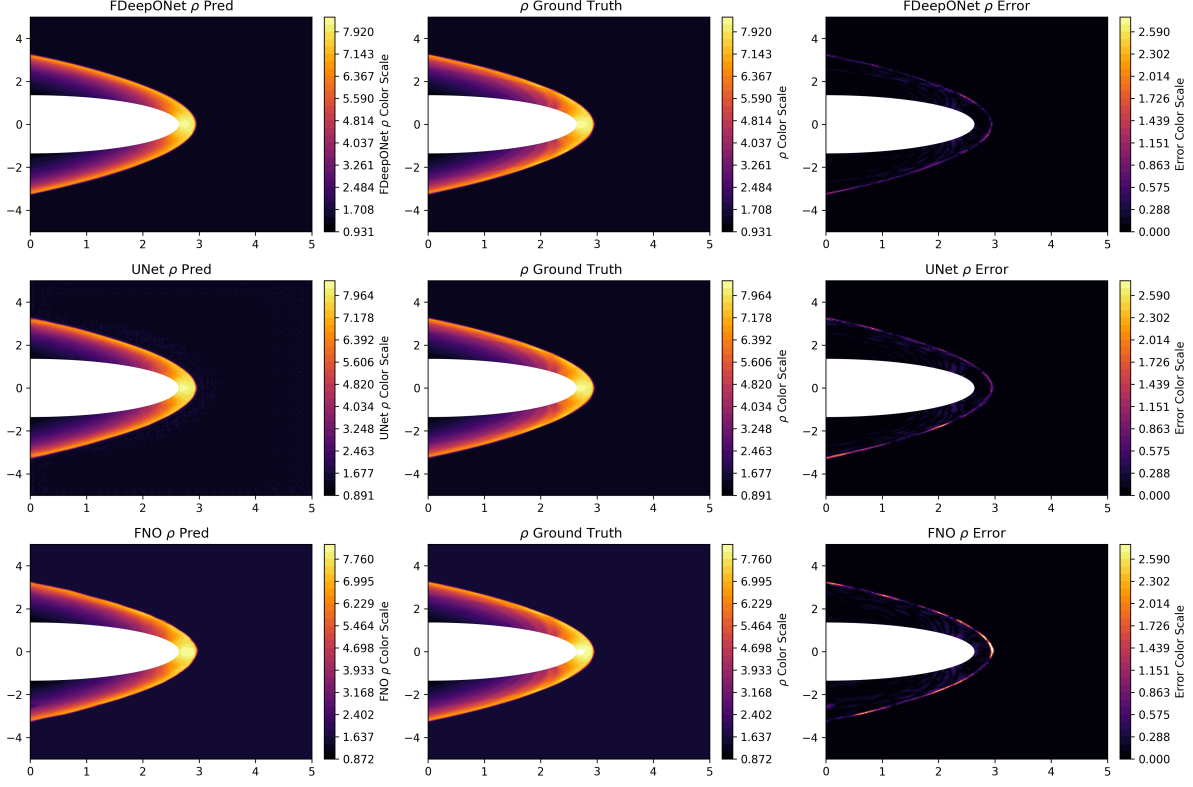




**Figure 3.10:** Prediction of Fusion-DeepONet for three unseen samples. Each row depicts  $u$  and  $v$  prediction, ground truth, and point-wise error for an unseen sample on uniform grid. The first two columns show the predicted  $u$  and the ground truth. The last two columns depict the predicted  $v$  and ground truth. Row 1: test case 1, Row 2: test case 5, Row 3: test case 8. See appendix A2.1 for the values of input parameters for cases 1,5 and 8.

and UNet in particular, since the high error region in FNO prediction has more thickness. Qualitatively, all the operators can predict the general shape of the solution. However, Fusion-DeepONet better predicts the bow shock edges surrounding the semi-ellipse, as shown by the point-wise error plot in the third column. The FNO prediction of the bow shock edge shows wiggles, and its point-wise error is the highest. The comparison of other variables, such as  $u$ ,  $v$ , and  $p$ , shows a similar conclusion as the density plots. Results from FNO show higher error values when compared to Fusion-DeepONet and U-Net for a regular grid setup for all output variables except for the  $v$  velocity component, where the error is the lowest. This may be due to using the same modes used to filter all four output variables during training simultaneously. The design parameters for the geometry were not used as inputs to the FNO model, which instead relies on the grayscale mask to map the shape of the geometry. Including the geometric parameters along with the node coordinates may improve FNO predictions; however, this would increase the memory footprint of the input data since each design parameter would have to be replicated across all nodes. Figure 3.12 depicts the MSE loss function value versus the Fusion and

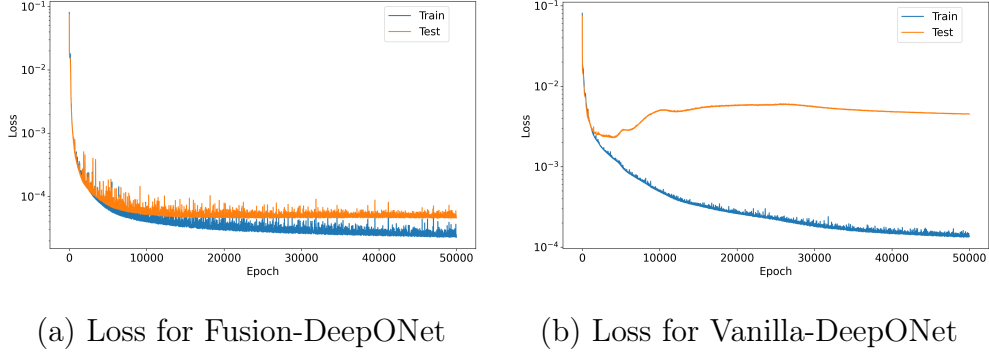
Vanilla-DeepONets training epoch. Fusion-DeepONet helps reduce over-fitting with the help of fusion connections and the conditioning of the trunk network. Therefore, the generalization error in Fusion-DeepONet is less than that of Vanilla-DeepONet.



**Figure 3.11:** Comparison of density prediction using Fusion-DeepONet, U-Net, and FNO neural operators for the unseen case 2 on uniform Cartesian grid. Each row corresponds to a specific neural operator. First row: Fusion-DeepONet, Second row: U-Net, and Third row: FNO. First column shows prediction by the neural operator and second column shows the ground truth with a similar color range as the prediction case. Third column shows absolute value of point-wise error. The color bar for the point-wise error is the same for various neural operators for comparison. See appendix A2.1 for value of input parameters of test case 2.

## Training on irregular grid

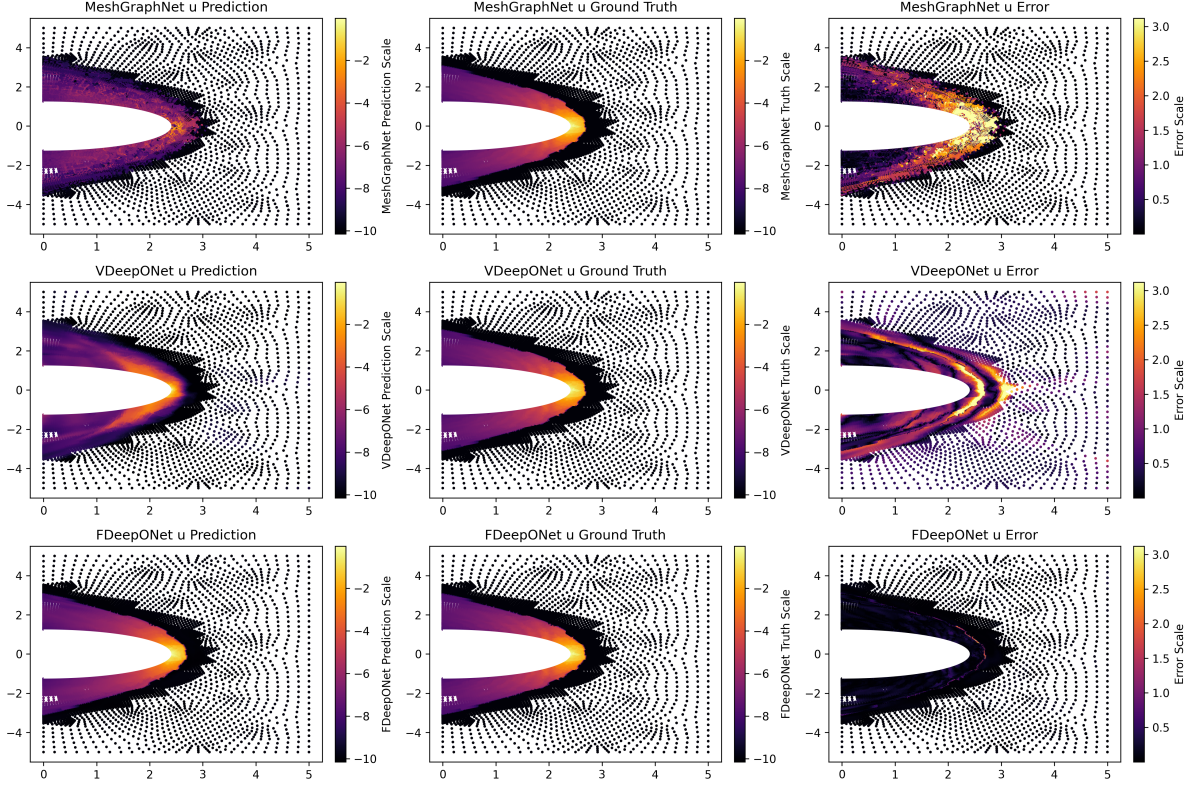
Prediction of a hypersonic field over an irregular grid poses a significant challenge to neural operators. In the literature, few neural operators can handle irregular mesh and sample-dependent grid points directly. This study uses the Vanilla-DeepONet and MeshGraphNet frameworks to evaluate Fusion-DeepONet performance on irregular grids. We modified the structure of the DeepONet framework into a unique conditioned neural



**Figure 3.12:** *Loss versus epochs for Fusion and Vanilla-DeepONets uniform grid.*

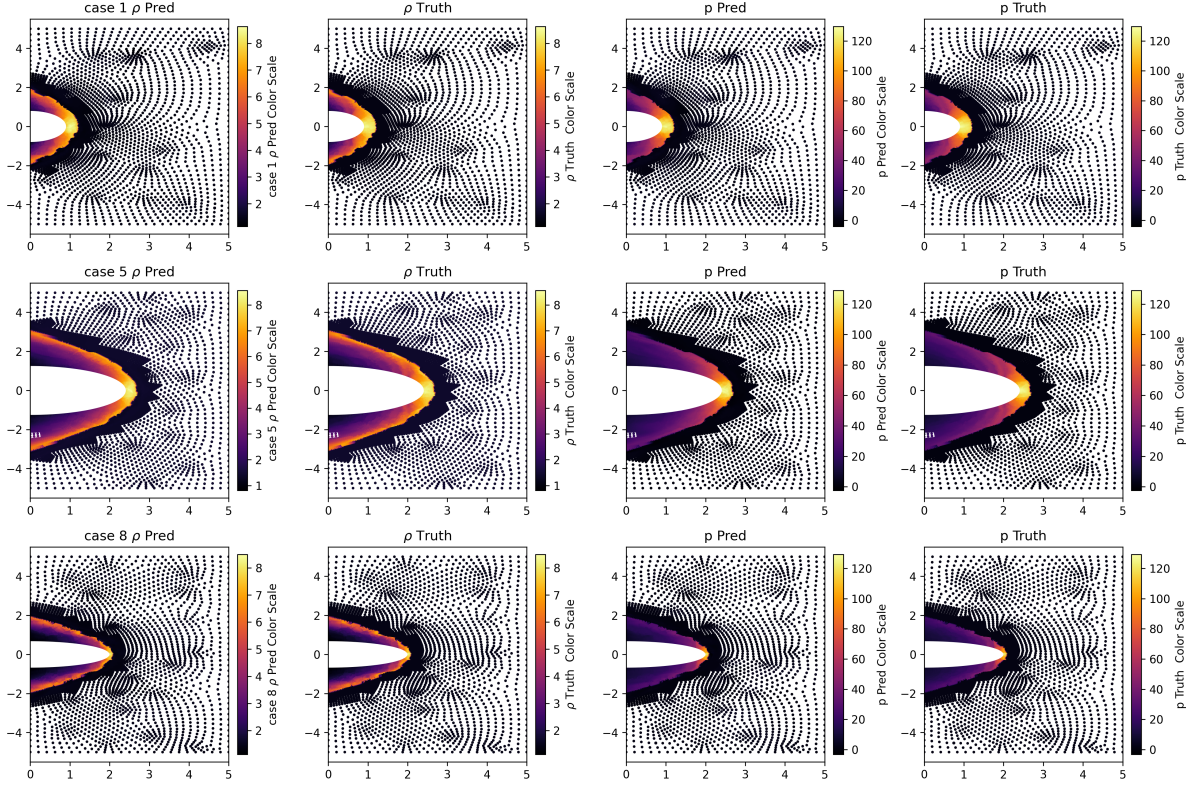
field that can build geometry-informed basis functions. The geometry-informed basis functions encode the grid points and geometry parameters and adapt to the hypersonic flow field simultaneously. MeshGraphNet utilizes message-passing to update node and edge feature embeddings using the initial nodal coordinates and Euclidean distance between the nodes as the node and edge features of the graph network. By utilizing the node coordinates with node connectivity across the whole domain, MeshGraphNet enables the creation of better feature embeddings that can act as basis functions for the nodal predictions. By predicting fields  $(\rho, u, v, p)$  defined as the output node features trained in a supervised manner, MeshGraphNet maps the structural features of the geometry to the resulting flow fields, thereby eliminating the need for using geometric parameters explicitly.

Figure 3.13 visually compares the predicted solution by Fusion-DeepONet, Vanilla-DeepONet, and MeshGraphNet operators. Fusion-DeepONet can accurately predict the  $u$  field without spurious oscillations or excessive diffusion of sharp interfaces in the solution. Fusion-DeepONet generalizes to various geometries without requiring the explicit input of the geometric boundary of the flying object. In a sense, it implicitly learns the location of the boundary by fusing the branch net information into the trunk network. MeshGraphNet performs the worst for the irregular grid problem and cannot predict the sharp edge of the bow shock or the solution profile between the semi-ellipse and the bow shock. The Fusion-DeepONet excels in learning non-linear geometry-dependent solutions on

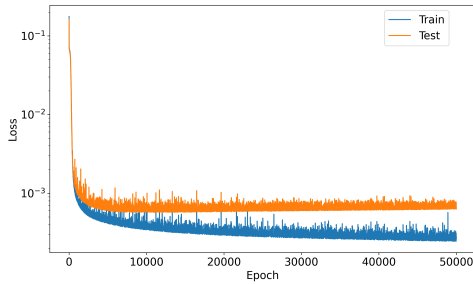


**Figure 3.13:** Comparison of  $u$  prediction using MeshGraphNet, Vanilla-DeepONet, and Fusion-DeepONet neural operators for the unseen case 5 on an irregular unstructured grid. Each row corresponds to a specific neural operator. First row: MeshGraphNet, Second row: Vanilla-DeepONet, and Third row: Fusion-DeepONet. First column shows prediction by the neural operator and second column shows the ground truth. Third column shows absolute value of point-wise error. The color bar for the point-wise error is the same for various neural operators for comparison. See appendix A2.1 for value of input parameters of test case 5.

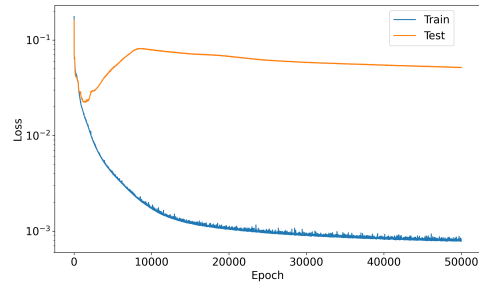
an irregular grid. Comparing the Vanilla-DeepONet versus the Fusion-DeepONet, we can deduce that the fusion of branch layers into the trunk network improves learning generalization over arbitrary grids. For completeness, we depict the effectiveness of Fusion-DeepONet in learning geometry-dependent hypersonic field for three unseen samples in Fig. 3.14. Figure 3.15 demonstrates the MSE loss function value for Fusion-DeepONet and Vanilla-DeepONet trained on cases with irregular unstructured grids. The generalization error for Vanilla-DeepONet is much larger than that of Fusion-DeepONet.



**Figure 3.14:** Prediction of Fusion-DeepONet for three unseen samples on irregular unstructured grids. Each row depicts  $\rho$  and  $p$  prediction and ground truth for an unseen sample. the first two columns show the predicted  $\rho$  and the ground truth. The last two columns depict the predicted  $p$  and ground truth. Row 1: case 1, Row 2: case 5, Row 3: case 8. See appendix A2.1 for the values of input parameters for test cases 1,5 and 8



(a) Loss for Fusion-DeepONet



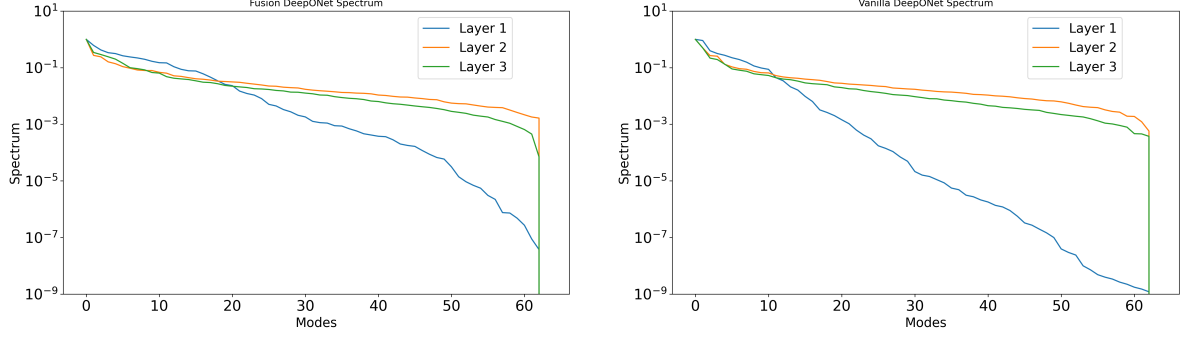
(b) Loss for Vanilla-DeepONet

**Figure 3.15:** Loss versus epochs for Fusion and Vanilla-DeepONets on irregular grid.

## Interpretation

This section analyzes the reasons for Fusion-DeepONet's good performance for geometry-dependent problems. Within the framework of the DeepONet, the trunk network encodes





(a) Energy Spectrum Fusion-DeepONet      (b) Energy Spectrum Vanilla-DeepONet

**Figure 3.16:** Energy spectrum for modes of trunk network hidden layers outputs trained with uniform grid training dataset for Vanilla and Fusion-DeepONet.

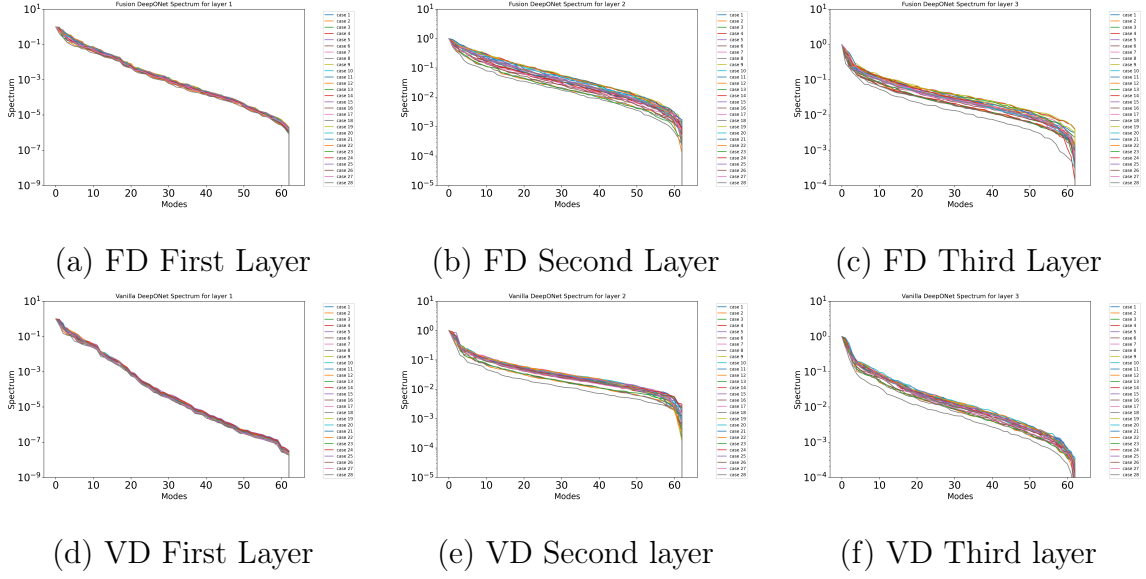
the coordinate points into a set of basis functions that are conditioned using the encoded output of the branch hidden layers. We borrow the analysis method of Peyvan *et al.* [8] to decompose the output of each hidden layer of the trunk network into interpretable entities. We performed this analysis on both uniform and irregular grids. Using the same set of grid points for all the training samples differentiates entirely from using individual discretization for each sample. We first analyze the trunk network of the Fusion-DeepONet and Vanilla-DeepONet in learning the uniform grid problem.

**Uniform grid:** We apply SVD on all the hidden layers of the trunk net to compute 64 eigenvalues corresponding to 64 eigenvectors. Therefore, we evaluate the energy spectrum for each of the 64 modes corresponding to the number of neurons of each hidden layer. Figure 3.16(a) and (b) compare the energy spectrum of the output of hidden layers in the trunk network of Fusion and Vanilla-DeepONets. According to the energy spectra of the first hidden layer output, we observe that the Fusion-DeepONet extracts more information both in the low and highest modes. As a result, the first layer collects high-frequency information due to the connection of the first layer of the branch to the last layer of the trunk network. Conditioning the output of the trunk network layers brings extra information from the geometric parameter encoding into the trunk network to modulate the data-driven basis functions.

**Irregular grid:** Interpreting the prediction of Fusion-DeepONet on an irregular grid is much more challenging than on a uniform grid. The trunk network must construct 64 basis functions corresponding to each training sample for each grid configuration. There exist 28 samples in the training dataset providing 28 different grid configurations. During the training of Fusion and Vanilla-DeepONets, the trunk network must encode information on both the solution structure and mesh point configurations and can generalize both the solution values and mesh point locations. As shown in Fig. 3.13, Vanilla-DeepONet struggles with predicting the flow field over the changing geometry, while the Fusion-DeepONet can accurately infer the solution. We now investigate the SVD of the output of the three hidden layers in the trunk network of Fusion and Vanilla-DeepONets. Figure 3.17(a)-(f) demonstrates the spectrum of the outputs of the hidden layer for 28 grid configurations. Each plot shows the spectrum of modes for each grid configuration in a particular color. Considering the spectrum on all the 28 grid configurations, the amount of energy captured over the entire modes by the first hidden layer of Fusion-DeepONet is much higher than the Vanilla-DeepONet, as seen in Fig. 3.17(a) and (d). Similarly, the second and the third hidden layers of the Fusion-DeepONet extract more information over the entire range of modes than the Vanilla-DeepONet. Comparing Fig. 3.17(b) and (e), we can observe that the curves corresponding to grid configurations are more scattered in Fusion-DeepONet, while for Vanilla-DeepONet, the curves are clustered together. This clustering of the spectrum for all the grid configurations can hint at the fact that the Vanilla-DeepONet cannot differentiate between various grid configurations and, in other words, cannot learn and generalize on the locations of the coordinate points.

## Error Distribution

We also compare the sample-wise errors for all test conditions to understand the spread and identify outliers that may skew the error results. Figure 3.18 shows a bar plot comparing the spread of individual sample errors for the four regular and irregular grid fields. We note that the Vanilla-DeepONet and MeshGraphNet show the highest mean



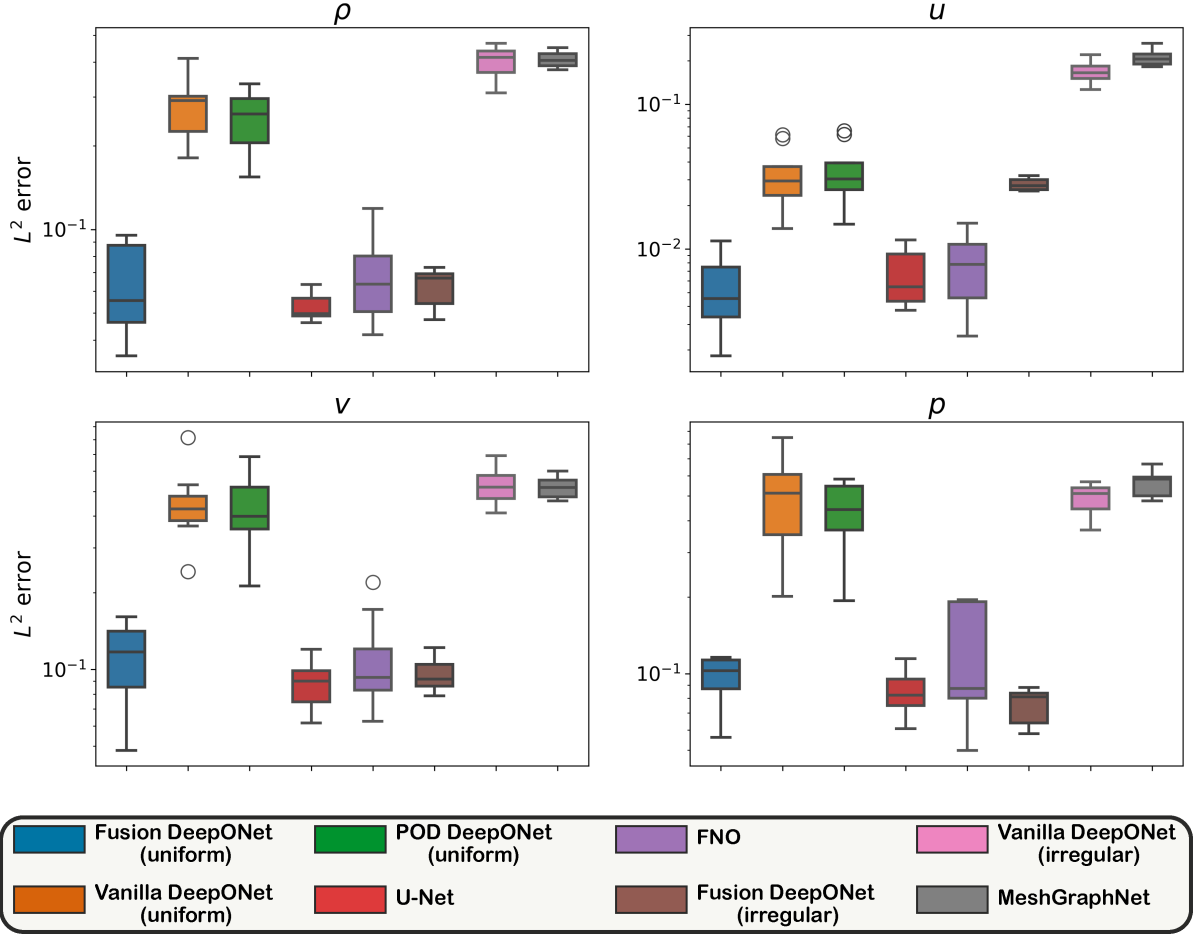
**Figure 3.17:** *Energy Spectrum of trunk network's hidden layers outputs of Fusion-DeepONet (FD) and Vanilla-DeepONet (VD) for irregular unstructured grid. Each curve represents the Eigen spectrum of extracted data on an individual grid in the training dataset.*

errors compared to Fusion-DeepONet, POD-DeepONet, U-Net, and FNO. U-Net and Fusion-DeepONet for regular grids show the lowest error values due to their ability to extract features across multiple scales. U-Net achieves the lowest mean error values on the regular grid for  $\rho$ ,  $v$ , and  $p$ , while Fusion-DeepONet performs better than its counterparts for the irregular grid.

### 3.5.2 Hypersonic viscous flow around reentry capsule

We develop this test case to assess the accuracy of high-speed flow surrogate models in predicting surface heat flux on reentry vehicles—a critical factor in designing thermal protection systems. Our goal is to demonstrate that Fusion-DeepONet can generate surrogates for geometry and varying flow conditions, such as Mach number. As a result, the capsule's shape can be optimized across the whole reentry trajectory, rather than at a single fixed Mach number. From 60 total simulation runs, we randomly allocated 48 cases for training and 12 for testing. Since we focus on heat-flux prediction, Fusion-DeepONet is configured to predict only the temperature field. The trunk and branch sub-networks comprise five hidden layers of 64 neurons each, followed by a 64-neuron linear output





**Figure 3.18:** Box plot showing sample-wise error comparison for all architectures and grid types. Vanilla-DeepONet and MeshGraphNet frameworks show highest sample-wise error distribution across both mesh types. Error values for  $u$ -velocity prediction show the lowest error across all the four outputs predicted in this study. U-Net and Fusion-DeepONet show better error values across all four output predictions. Only the Fusion-DeepONet provides a good error estimate across both regular and irregular grid setup.

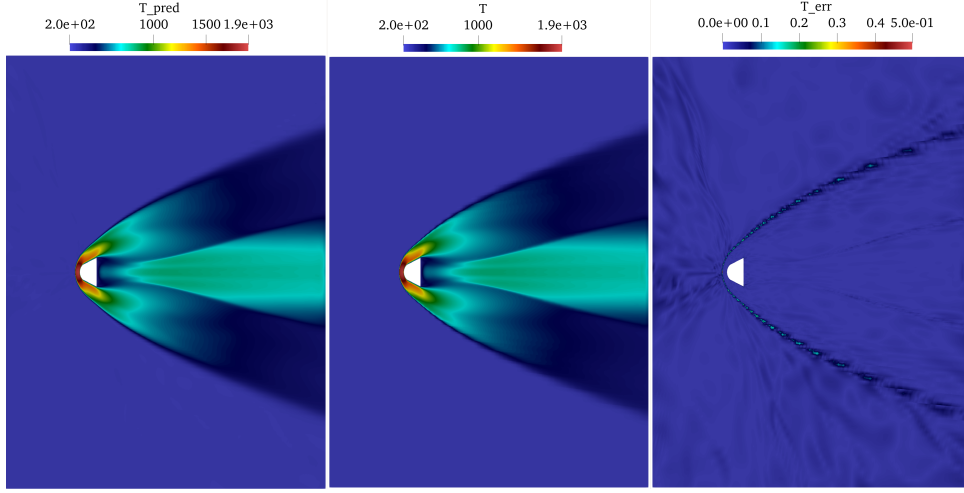
layer. We employ the Rowdy activation function in all hidden layers and train the model using the Adam optimizer for 100,000 epochs on an NVIDIA L40S GPU. Each sample contains 77,171 unique solution points for learning the temperature distribution.

To predict the heat flux, the temperature surrogate model must accurately predict the temperature derivative with respect to the  $x$  and  $y$  coordinates. The surrogate model generated using Fusion-DeepONet is a continuous function that takes in the geometric parameters, the Mach number, plus the  $x$  and  $y$  coordinates, and generates the temperature predictions. Since we generate a continuous function, we can efficiently

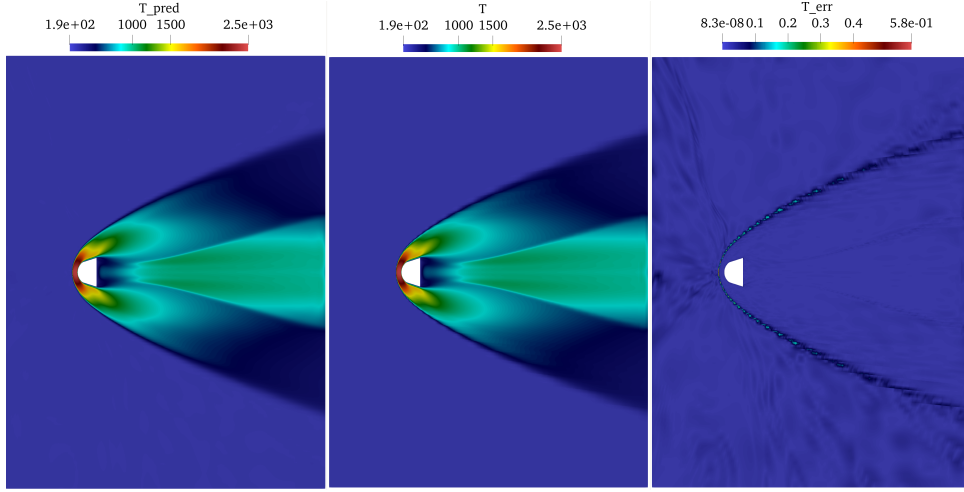
compute its derivative with respect to the inputs, including geometric parameters, Mach number, and coordinates, using automatic differentiation. Ultimately, we compute the heat flux by applying automatic differentiation on the trained Fusion-DeepONet to compute the heat flux on the surfaces of the reentry vehicle geometries shown in Fig 3.6. Fusion-DeepONet is trained employing two loss functions. First, we use the mean squared error (MSE) loss function constructed using the predicted temperature and the ground truth. Second, we use the loss functions defined in Eq. (3.4.6), which is a combination of MSE and a loss term for the derivative of the temperature field.

Our objective is to develop a surrogate model for the temperature field around a hypersonic reentry capsule that can accurately predict temperature derivatives while trained on temperature values. A key challenge is enhancing the derivative prediction capability of Fusion-DeepONet. Since we assume no access to the connectivity matrix of the solution points in the computational domain, we avoid using the derivative matrix from the DG solver that generated the training data. Instead, we aim to create an effective method even when working with unstructured point clouds lacking mesh connectivity. To address this, we explore several strategies to enrich Fusion-DeepONet with derivative information from the temperature field.

Figure 3.19 presents the temperature field predictions made by Fusion-DeepONet. In each set of figures, the left plot illustrates the predicted temperature field, the middle plot represents the ground truth, and the right plot shows the point-wise prediction error calculated as  $|T_{\text{pred}} - T|$ . The results demonstrate that Fusion-DeepONet achieves highly accurate temperature predictions for previously unseen cases, highlighting its excellent generalization capability across varying geometries and Mach numbers. The point-wise errors remain small throughout most of the domain, although the maximum errors are observed near the bow shock region. The average relative  $L_2$  norm of the temperature error across the 12 test cases is approximately 0.98%. However, as discussed later, despite the temperature field being predicted with less than 1% error, significant errors emerge when computing its spatial derivatives with respect to the  $x$  and  $y$  coordinates.



(a) Temperature field case 4  $(R, \alpha, Ma) = (14.59, 44.40, 6.24)$



(b) Temperature field case 11  $(R, \alpha, Ma) = (17.25, 54.19, 7.36)$

**Figure 3.19:** Two unseen test cases of flow field around hypersonic reentry vehicle predicted by Fusion-DeepONet. (a) Temperature case 4, (b) Temperature case 11. The Fusion-DeepONet was trained using 48 samples and tested for 12 unseen samples. The loss function is MSE of temperature field. from the left first contour plot is prediction, second plot is ground truth, and third plot is the point-wise error defined as  $T_{err} = |T_{pred} - T|$ , See appendix. A2.1 for the values of input parameters for test cases 4 and 11.

Now we focus on the correct prediction of the heat flux on the surface of the reentry vehicle. The heat flux can be determined using the following

$$Q = \int_{\partial\Omega} -\kappa \nabla T \cdot \vec{n} ds \quad (3.5.1)$$

where  $\nabla T = (\partial T / \partial x, \partial T / \partial y)$ ,  $\vec{n}$  is the normal vector to the line segment  $ds$ .  $\partial\Omega$  denotes

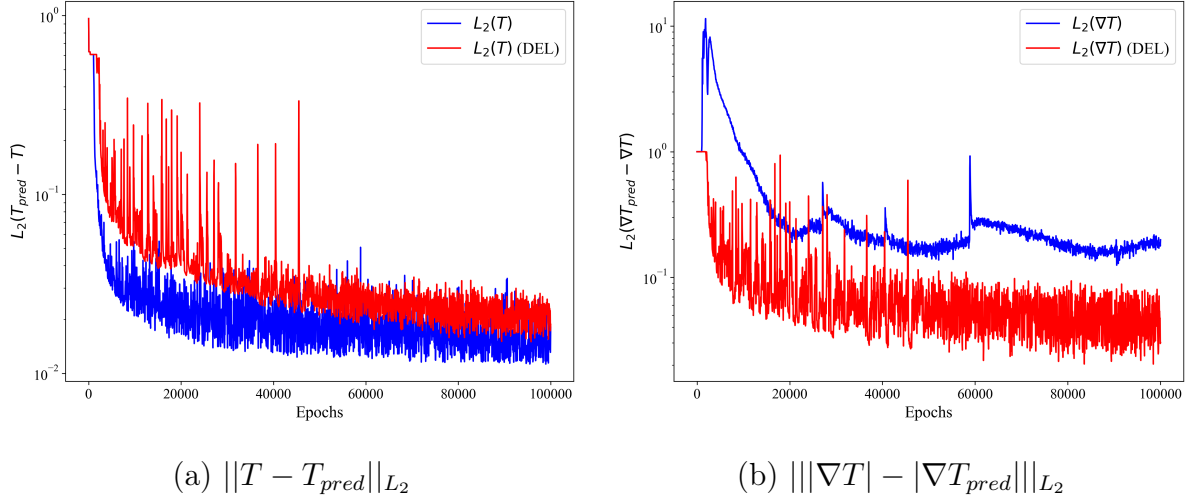
the surfaces of the reentry capsule. The surfaces of the reentry capsule are segmented using the sides of quadrilateral elements placed on the boundary of the capsule. According to the discretization of the DGSEM solver, on each line segment, we have  $P + 1$  nodes which we use to approximate the integral of Eq. (3.5.1). In the simulations, we employed  $P = 3$ , which results in four points per line segment. For each line segment, we used the trapezoid rule using the four nodes, the normal vector values to the line segment, and the value of  $\nabla T$  computed at four nodes to compute the heat flux. After the computation of heat flux of line segments, we sum the heat flux of the line segments on the surface of the reentry capsule to compute the total heat flux.

### Heat flux prediction

In Section 3.4.2, we introduced two methods to enhance the accuracy of derivative predictions by Fusion-DeepONet and compare them with automatic differentiation approach:

1. Discrete Derivative (DD) Method: Use the discrete derivative operator defined in Eq.(3.4.8) to compute derivatives of both the Fusion-DeepONet predictions and the ground truth, as incorporated in the derivative loss term in Eq.(3.4.6).
2. Least Squares Differentiation (LSD) Method: Estimate directional derivatives of both the Fusion-DeepONet predictions and the ground truth using the LSD method, and include them in the derivative loss term in Eq. (3.4.6).

We further investigate their accuracy in predicting the temperature field and the temperature derivative without explicitly training DeepONet for the derivative field. Here, we clarify that the ground truth data is generated using the discontinuous Galerkin code, which produces a discontinuous solution at the interfaces of the quadrilateral elements. Therefore, when we compute the derivative of this temperature field even in the ground truth, we obtain discrete derivative contours across elements. To diminish the discontinuous jumps in the temperature derivative for the ground truth, we use spectral element interpolation to interpolate the solution on  $P = 3$  elements ( $4 \times 4$  nodes in each quad) to



**Figure 3.20:** (a)  $L^2$  norm of temperature predictions for testing dataset during the training of Fusion-DeepONet (b)  $L^2$  norm of temperature gradient magnitude computed using least square differentiation operator for testing dataset during the training of Fusion-DeepONet. (DEL) refers to derivative enhanced loss function.

$P = 7$  elements ( $8 \times 8$  nodes in each quad). Therefore, we generate a higher resolution grid without simulating the cases using Trixi.jl for higher  $P$ . During training, we employ  $P = 3$  resolution coordinates, while for inference, we employ both  $P = 7$  and  $P = 3$  for heat flux predictions.

In Fig. 3.19, we show that Fusion-DeepONet can precisely learn the temperature field. However, predicting the temperature field accurately does not guarantee the accuracy of  $\partial T / \partial x$  and  $\partial T / \partial y$ . Figure 3.20 (a) compares the relative  $L_2$  norm of temperature prediction error for the testing dataset during training for the MSE loss function and the derivative-enhanced loss function (DEL). Moreover, Fig. 3.20 (b) compares the trend of the  $L_2$  norm of the magnitude of the temperature gradient computed using the least squares differentiation operator. According to Fig. 3.20, the DEL loss function slightly deteriorates the temperature prediction while improving the temperature gradient prediction. We conclude that integrating a derivative loss term can enhance the prediction of the field gradient and, consequently, the heat flux.

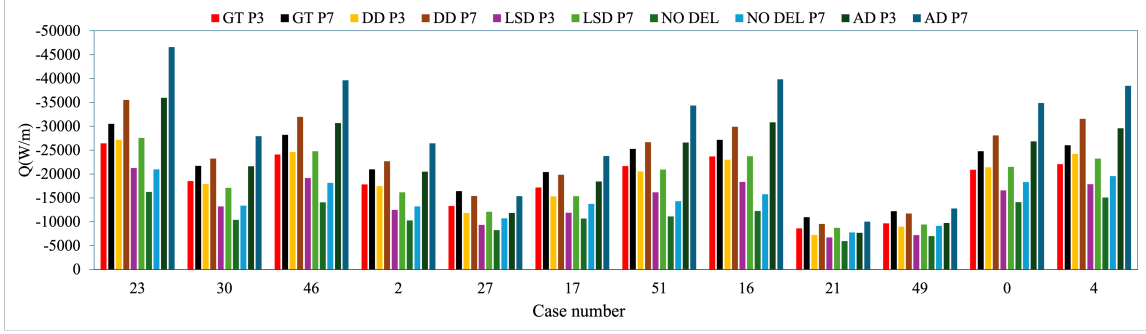
The derivative loss term can be implemented using three distinct approaches. Table 3.3 compares the accuracy of heat flux and temperature predictions obtained using various

**Table 3.3:** Heat flux prediction accuracy over the unseen dataset using various derivative operators.  $\%E$  is average relative error of heat flux predictions over the entire testing dataset.  $\%L^2(T)$  is the relative  $L^2$  norm of temperature field for all the test cases. The temperature derivative computation during the inference is performed using automatic differentiation for all the cases.

| Approach | $\%E(Q), P = 3$ | $\%E(Q), P = 7$ | $\%L^2(T), P = 3$ | $\%L^2(T), P = 7$ | time(h) |
|----------|-----------------|-----------------|-------------------|-------------------|---------|
| NO DEL   | 38.49           | 33.36           | 0.98              | 0.95              | 8.67    |
| AD       | 20.06           | 29.55           | 3.49              | 3.46              | 22.25   |
| LSD      | 24.54           | 17.80           | 1.30              | 1.33              | 9.39    |
| DD       | <b>6.28</b>     | <b>10.09</b>    | 1.34              | 1.31              | 8.00    |

loss functions. We computed the heat flux predictions on a coarse grid ( $P = 3$ ) and a fine grid ( $P = 7$ ). Our results indicate that using the discrete derivative (DD) operator to construct the derivative-enhanced loss (DEL) term achieves the highest accuracy at the lowest computational cost. However, because the DD approach uses a discrete derivative operator, the neural operator does not exhibit super-resolution capabilities. Conversely, the least squares derivative (LSD) and automatic differentiation (AD) methods enforce resolution-independent learning of temperature derivatives, demonstrating that employing a higher-resolution mesh significantly improves heat flux prediction accuracy. Among all the derivative-enhanced methods, using AD is the most computationally expensive model. While AD improves heat flux prediction on  $P = 3$  grid resolution, it deteriorates the accuracy of temperature prediction more than the other approaches.

The last column of Table 3.3 highlights the computational cost associated with training Fusion-DeepONet using different loss functions over 100,000 epochs. The DD operator presents the lowest computational expense. Based on the data presented in Table 3.3, we conclude that the optimal choice of derivative loss term depends on the grid resolution used for data generation: DD is preferable for coarse-resolution meshes, while LSD provides superior accuracy for high-resolution meshes. Figure 3.21 illustrates heat flux predictions for various test cases using different loss functions. Fusion-DeepONet was trained on the



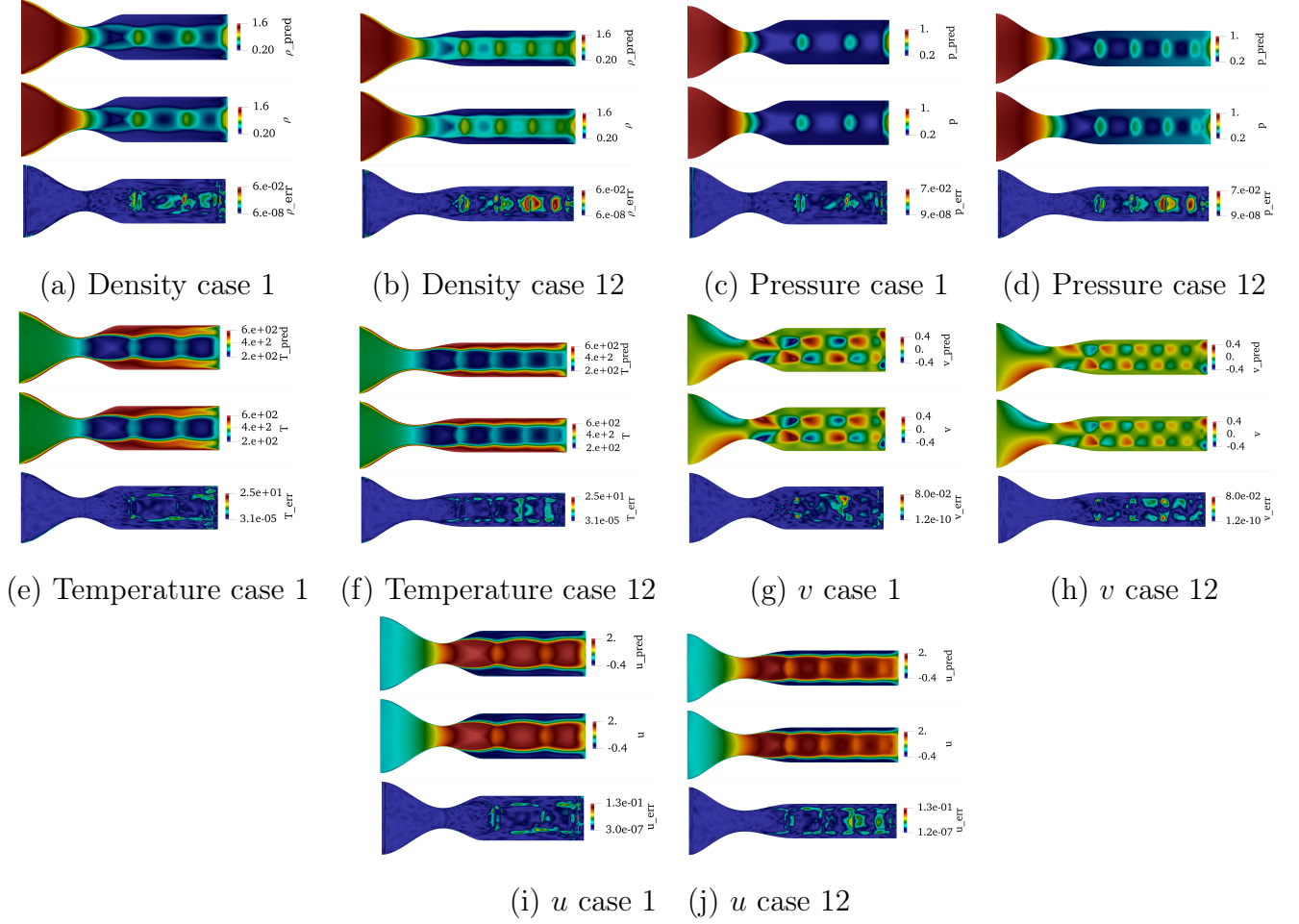
**Figure 3.21:** Heat flux prediction using various loss functions for test samples. *GT*: Ground Truth, *DD*: Discrete Derivative operator, *LSD*: Least Square Differentiation, *AD*: Automatic Differentiation, *NO DEL*: No derivative term in the loss function, *P3*: refers to coarse mesh with polynomial order 3, *P7*: refers to fine mesh with polynomial order 7.

$P = 3$  grid and evaluated on both  $P = 3$  and  $P = 7$  grids. According to Fig. 3.21, the discrete derivative loss consistently delivers the most accurate heat flux predictions across all tested cases.

### 3.5.3 Viscous flow inside convergent-divergent nozzle

In this example, we learn a complex internal flow field inside a converging-diverging nozzle. Fusion-DeepONet learns a mapping from the shape of the nozzle to a steady state flow that includes density,  $u$ ,  $v$ , pressure, and temperature. We split 60 cases into 42 training and 18 testing samples in this example. The Fusion-DeepONet architecture is designed to have 128 neurons in five hidden layers for the branch and trunk sub-networks. We use a single trunk network for all the trained variables (density,  $u$ ,  $v$ , pressure, and temperature). To predict five variables, we employ  $5 \times 128$  neurons in the last linear layer of the branch network. We train the model for 100k epochs and select the best model among the saving checkpoints based on the best relative  $L^2$  norm of the testing samples' error. We employ 16,751 unique coordinate points extracted from Trixi.jl results for training. For this problem, we did not employ the solution gradient loss terms to enhance the prediction of the solution derivative.

Table 3.4 compares the accuracy for an unseen testing dataset for five primitive variables predicted by Fusion-DeepONet and MeshGraphNet. The training of Fusion-DeepONet is



**Figure 3.22:** Two unseen test cases of converging-diverging nozzle flow field predicted by Fusion-DeepONet. All the variables are non-dimensional except temperature which is reported in Kelvin unit (K). (a) Density case 1, (b) Density case 12, (c) Pressure case 1, (d) Pressure case 12, (e) Temperature case 1, (f) Temperature case 12, (g)  $v$  case 1, (h)  $v$  case 12, (i)  $u$  case 1, (j)  $u$  case 12. The Fusion-DeepONet was trained using 42 samples and tested for 18 unseen samples. In each sub-figure, from the top, the first plot shows prediction, second shows ground truth, and third shows point-wise error defined as  $E = |\phi_{pred} - \phi|$  where  $\phi$  denotes the primitive variables. See appendix. A2.1 for the values of input parameters for test cases 1 and 12.

performed using the JAX backend on the NVIDIA H100 GPU, while we used NVIDIA 3090 GeForce RTX using PyTorch for MeshGraphNet. Fusion-DeepONet can accurately learn the primitive variables for nozzle shape surrogate modeling better than MeshGraphNet for all the primitive variables. The precision of the Fusion-DeepONet deteriorates when inferring the  $v$  component. According to Figs. 3.22(a)-(f) and (i) and (j), we can observe that the solution structure for density, pressure,  $u$ , and temperature shows some degree of similarity. However, the solution structure of  $v$  entirely differs from the rest. We mentioned



**Table 3.4:** Comparison of prediction error for the converging-diverging nozzle problem,  $\#P$  refers to number of trainable parameters. The  $L^2$  norm reported is the average of relative  $L^2$  norm across 18 unseen samples.

| Framework       | $\#P$   | $\%L^2(\rho)$ | $\%L^2(u)$  | $\%L^2(v)$   | $\%L^2(p)$  | $\%L^2(T)$  | time(sec) |
|-----------------|---------|---------------|-------------|--------------|-------------|-------------|-----------|
| Fusion-DeepONet | 298,182 | <b>1.29</b>   | <b>2.41</b> | <b>11.26</b> | <b>1.39</b> | <b>0.97</b> | 5,569.5   |
| MeshGraphNet    | 277,061 | 4.00          | 7.25        | 23.33        | 3.29        | 3.66        | 13,000    |

previously that a single trunk sub-network generates the basis functions for predicting five variables, hence a higher error for  $v$ . Figure 3.22 shows that Fusion-DeepONet can predict the flow field near the boundaries, specifically near the wall boundary conditions. Moreover, comparing the flow field of Case 1 with Case 2, we observe that the flow field significantly varies since the geometry of the nozzle changes.

### 3.6 Summary

In this study, we propose three challenging problems for developing a neural operator-based framework for geometry-dependent hypersonic and supersonic flow problems. We use these three problems to evaluate the newly developed Fusion-DeepONet in learning high-speed flow fields for varying geometries. First, we compare several state-of-the-art neural operators, including parameter-conditioned U-Net, DeepONet, FNO, and MeshGraphNet, with our Fusion-DeepONet in learning a challenging geometry-dependent hypersonic inviscid flow around blunt bodies for uniform Cartesian and irregular grids. For the training dataset, we deliberately employed a limited number of samples. The scarce data set includes the density, x-direction velocity, y-direction velocity, and pressure of the hypersonic flow around a semi-ellipse that is parametrized using values of the minor and major axes. The solution involves a highly non-linear field with strong shocks and high-gradient regions.

Using 28 training samples, we first created a Cartesian uniform grid and interpolated all the samples into the Cartesian grid. This allows us to evaluate and compare different

operator models that work on uniform grid. U-Net and Fusion-DeepONet outperformed other neural operators in generalization and prediction accuracy. However, Fusion-DeepONet achieved the same accuracy as U-Net using approximately 200 times fewer parameters than the U-Net framework. Moreover, the training time for Fusion-DeepONet is less than that of U-Net. The primary focus of this study was to develop a framework that can input arbitrary grids and generalize using a limited number of samples. Thereafter, we employ this Fusion-DeepONet to use unstructured and irregular grids for various training samples and predict the hypersonic flow field with strong discontinuities. We employed Vanilla-DeepONet and MeshGraphNet to learn the solution field on an irregular, unstructured grid for comparison. They both performed far worse than Fusion-DeepONet and do not generalize to the irregular grid solution field. We then analyzed the reason for improved performance of Fusion-DeepONet using SVD decomposition of its trained trunk network. Using the decomposition of hidden-layer outputs, we observed that the Fusion-DeepONet approach can extract the maximum amount of information from the training dataset for the low and high modes. The decomposition of the trained trunk network using irregular grids shows that the Fusion-DeepONet can learn the grid-dependent information far better than the Vanilla and POD-DeepONet at their respective lowest generalization error.

Next, we learn viscous hypersonic flow around a reentry capsule, parameterized by two geometric variables and one flow condition (Mach number). The primary quantity of interest in this context is wall heat flux. To enable accurate heat flux prediction, we design a surrogate model that predicts the temperature field, from which surface heat flux can be computed. We augment the mean squared error loss with a derivative-informed term to enhance the model’s ability to capture temperature gradients. This additional term improves Fusion-DeepONet’s accuracy in predicting temperature derivatives at the boundaries, which is critical for accurate heat flux estimation. We demonstrate that even highly accurate temperature predictions do not necessarily yield accurate heat fluxes unless the model is enhanced to capture temperature gradients. Incorporating the

derivative-based loss allows Fusion-DeepONet to serve as a robust surrogate model for hypersonic heat flux prediction, suitable for early-stage aerodynamic design. Moreover, we demonstrate that Fusion-DeepONet can construct a surrogate model dependent on both geometry and Mach number, enabling optimization of the reentry capsule geometry along the entire reentry trajectory across varying Mach numbers.

Finally, we apply Fusion-DeepONet to learn supersonic internal flow under varying geometries. A converging-diverging nozzle connected to an isolator duct is parameterized using three geometric variables. A single Fusion-DeepONet model is trained to simultaneously predict density, x-velocity, y-velocity, pressure, and temperature fields. For this complex scenario, Fusion-DeepONet significantly outperforms MeshGraphNet regarding prediction accuracy.

In future, we intend to apply our Fusion-DeepONet framework to learn hypersonic flow fields involving real chemistry and extend it further to learn a mapping from non-parametric geometries to time-dependent hypersonic flows in 2D and 3D spatial configurations.

# CHAPTER 4

## Learning Nonlinear Responses in PET Bottle Buckling with a Hybrid DeepONet–Transolver Framework

### 4.1 Introduction

Developing neural surrogates to learn functional mappings for PDE problems defined on non-Euclidean domains has recently gained significant research attention. Traditional machine learning tasks in fields such as computer vision and signal processing benefit from statistical properties like stationarity and compositionality, which naturally arise from the Euclidean structure of the data [114]. By contrast, engineering design problems frequently involve non-Euclidean data, where characteristics such as common coordinate system or shift invariance do not exist. Consequently, standard machine learning architectures such as Convolutional Neural Networks (CNNs) cannot be directly applied to such problems. An equally important challenge lies in the effective representation of non-Euclidean geometry data, as it plays a central role in determining the quality of the input–output mapping. Common strategies for representating such geometries include voxels, point clouds, meshes, zero level-set function such as Signed Distance Fields (SDFs), and parametric encoding,

each offering distinct advantages and limitations [115].

Operator networks, such as Deep Operator Network (DeepONet) [15], Fourier Neural Operator (FNO) [116], Wavelet Neural Operator (WNO) [87], among others, have emerged as powerful tools for neural PDE solving tasks. These approaches offer key advantages over conventional machine learning frameworks, including discretization invariance and improved generalizability. While these methods have demonstrated promising results across a range of applications, their applicability to learn PDE solutions consistently across multiple geometric domains remains limited. As a result, growing research efforts are directed towards developing *geometry-aware* neural surrogates that can effectively capture the mapping of PDE solutions over diverse geometries.

In this work, we focus on the problem of PET bottle buckling, an important task in computational mechanics and packaging design. Traditionally, this problem is addressed using finite element analysis (FEA), where each candidate geometry must be meshed and solved under prescribed loading conditions. While accurate, this approach is computationally intensive, particularly when large design spaces need to be explored, such as in optimization or uncertainty quantification studies. Evaluating hundreds or thousands of bottle designs, each requiring a nonlinear FEA simulation with potentially thousands (or even millions) of degrees of freedom, is prohibitively expensive and restricts the pace of design iteration. To overcome these limitations, we propose a *geometry-aware* neural surrogate that learns the mapping from bottle geometry to key mechanical responses, including displacement fields and time-dependent reaction forces. Specifically, we extend the Transolver framework [26] by integrating it with DeepONet, enabling simultaneous prediction of spatial and temporal responses. While Transolver effectively maps geometric features to nodal displacements, it is inherently restricted to node-based feature learning and lacks the ability to predict time-dependent outputs. By integrating Transolver with DeepONet, we address this limitation and design a hybrid DeepONet-Transolver framework for handling multi-task learning in bottle buckling analysis.

The key contributions of this work are:

- **Hybrid framework:** We introduce a novel integration of Transolver with DeepONet to create a geometry aware neural surrogate for bottle buckling analysis in computational mechanics and packaging design. We evaluate our framework on simulation data generated with Abaqus across novel and realistic bottle shapes with varying complexity. This framework uses attention mechanism for capturing short and long range geometry interactions which enables accurate prediction across multi-scale geometric features.
- **Multi-tasking approach:** The hybrid Transolver–DeepONet framework that extends the Transolver’s capability from learning spatial mappings alone to simultaneous prediction of nodal displacements and time-dependent reaction forces, making it applicable to broader classes of multi-task problems in computational mechanics. Our framework maps the original nodal points defining the geometry to respective nodal displacements that result due to top load compression on bottles. This helps in estimating the stiffness for different bottle designs under top compressive load. As an additional task, we predict the reaction force exerted at the top cap for predicting the potential onset of buckling for different bottle shapes. Both these quantities of interest are critical for bottle design.
- **Non-parametric geometries:** Our proposed approach utilizes non-parametric geometry data for learning functional mappings and hence, is readily applicable to industrial settings where parametric design data may not be available.

## 4.2 Background and related works

In this section, we review recent advances in *geometry-aware* neural methods, organizing them according to the underlying geometry representation. In particular, we highlight approaches based on meshes, parametric representations, and point clouds, as these formats have emerged as versatile and widely adopted choices for non-Euclidean geometries. Their

ability to capture complex geometrical features while balancing efficiency and accuracy makes them especially well-suited for the development of neural surrogates.

### **Methods using mesh-based representation**

Mesh-based representations are among the most widely used in engineering and scientific applications, and consequently, several neural surrogates have been developed to leverage this format. Geo-FNO [117] extends the FNO framework to irregular meshes by introducing a mapping that converts irregular grids to regular grids before applying Fourier feature learning. While this ensures computational efficiency, it requires additional overhead for grid mapping, which becomes particularly challenging for complex engineering geometries. MeshGraphNets use Graph Convolutions to map the node and edge features to the PDE solution, exploiting the inherent connectivity information of meshes. However, graph-based methods are limited by their efficacy and effectiveness in learning multi-scale and long distance geometric relationships to physical quantities. X-MeshGraphNet [24] improves the scalability of graph-based approach by partitioning large graphs into smaller sub-graphs and subsequently applying an aggregation technique to recover global solutions. This method has been successfully demonstrated for predicting aerodynamic pressure distributions and wall shear stress across 500 parametrically morphed car geometries in the DrivaerML dataset [118].

### **Methods using parametric representation**

Parametric representations provide a compressed, scalar encoding of geometric variations and have been used in geometry-based deep learning recently. For instance, [4] used a DeepONet framework to learn flow fields around different airfoil designs using two shape-defining parameters as inputs to the Branch network. In [22], geometric parameters were used to model the hypersonic flow field around two-dimensional objects to simulate capsule re-entry from outer space. Similarly, [119] utilized geometric parameters for predicting the temperature distribution in multiple three-dimensional conductive plates. Geom-DeepONet [120] combined parametric representation with applied load vector to

predict stress fields in simple beam designs. Despite their success, these methods are limited to parametric geometries and cannot handle non-parametric designs where the key differentiating features are unknown. This limitation is significant in engineering practice, where geometries are often represented in generic formats (e.g., STEP or IGES) to ensure cross-platform compatibility and protect intellectual property.

### **Methods using point cloud representation**

Point clouds provide a memory-efficient representation of geometry. Although unstructured, the spatial arrangement of points can be exploited to encode geometric information in neural networks. Several prior works, such as [121, 122, 123, 124] have been proposed to extract coordinate-based features from point clouds for computer graphic tasks such as classification and segmentation. More recently, these techniques have been adapted to develop neural surrogates for scientific regression tasks. For instance, [52] proposed a DeepONet-based framework for predicting solution to Poisson’s equation across varying PDE parameters, boundary conditions, forcing functions, and multiple convex polygon geometries. Drawing inspiration from PointNet, [125] introduced a DeepONet model that maps non-parametric bracket shapes, represented as point clouds, and variable loading conditions to stress fields resulting from linear elasticity. GINO [126] extended the FNO framework to arbitrary grids by transforming point clouds to a uniform grid using a connectivity-based approach with a ball  $B_r(x)$  around each grid point  $x$ , enabling discretization-invariant predictions. However, GINO incurs substantial pre-processing overhead, especially when applied to large domains.

CORAL [41] employs an encoder-decoder architecture, where the encoder produces a latent vector for input functions and measurement points via Implicit Neural Representation (INR) and the decoder maps this latent representation to outputs on different grids. CORAL demonstrated its effectiveness on a range of problems such as flow around two-dimensional airfoil designs, incompressible flow through pipes, and stress fields in a hyper-elastic material. Its key limitation is the requirement of an average deformation



reference mesh with the same nodal points as the target mesh, which is often infeasible in practical engineering settings. DOMINO [127] addressed the need to capture finer, local geometric details using a sequence of point convolution layers to project geometry onto the whole computational domain. While this framework was successful in predicting aerodynamic flow around complex car geometries, the model exhibited relative errors of  $\approx 20\%$  and requires significant pre-processing overhead. Similarly, [128] proposed a physics-informed DeepONet for multi-geometry, multi-parameter problems such as Darcy flow and linear elasticity. This approach encodes geometric boundaries using point-wise MLPs, which becomes computationally expensive for large-scale problems. Also, determining the boundaries in a complex three-dimensional shape is a challenging task, especially for geometries with fine feature variation between them.

Recently, Transformers [129] with their attention-based architectures, have opened a promising direction for developing *geometry-aware* neural surrogates for scientific machine learning. For instance, GNOT [130] introduced a heterogeneous linear attention block to mitigate the quadratic computational cost of standard attention mechanism, enabling the integration of multiple inputs into a unified attention score. In addition, a gating mechanism was proposed to enhance scalability for large-scale problems. GNOT demonstrated strong performance across diverse applications, including transonic flow over different airfoil designs, linear elasticity in varying domains, and multi-cavity flow prediction. Transolver [26] and its extension Transolver++ [131], provide an alternative strategy for multi-geometry problems. Prior point cloud-based frameworks typically rely on MLPs and pooling layers to encode geometry, which often results in information loss - particularly when fine geometric variations are critical for design analysis. In contrast, Transolver uses physically-consistent tokens of spatial coordinates and employs point-wise spatial outputs as supervised signals to align token embeddings through local attention. This approach removes the need for a separate geometry encoder while improving predictive accuracy by leveraging attention across physically-consistent point features. Such attention-based methods have shown excellent scalability and robustness in

large scale applications, such as DrivaerNet++ [132]. Other works based on Transformer attention such as Latent Neural Operator (LNO) [133] have also shown promising results in *geometry-aware* problems.

In this study, our objective is to leverage the local feature extraction capabilities of Transolver framework to learn the buckling characteristics of multiple packaging bottle designs under top compressive load. Specifically, for a set of non-parametric domain geometries  $\Omega_i$ , we aim to approximate the operator  $\mathcal{G}$

$$\mathcal{G} : \{\Omega_i\} \longrightarrow u_i(\mathbf{x}), \quad \mathbf{x} \in \Omega_i, \quad \Omega_i \in \mathbf{\Omega} \subseteq \mathbb{R}^3, \quad (4.2.1)$$

where  $u_i(\mathbf{x})$  denotes the solution at the set of collocation points,  $\{x_j\}_{j=1}^{n_i}$ , that define the geometry  $\Omega_i$ . In addition to spatial field prediction, we also seek to predict the transient reaction force  $F_R$ , experienced during bottle compression, which is a measure of the bottle stiffness and its buckling characteristics. Since Transolver only predicts spatial fields at collocation points  $x_j$ , we integrate a DeepONet framework to concurrently predict reaction force  $F_R(t)$ . Specifically, DeepONet learns the operator

$$\mathcal{H}_\Theta : \mathcal{K}(\Omega_i) \times T \longrightarrow \mathbb{R}, \quad (k, t) \mapsto F_R(t), t \in [0, 1], k \in \mathbb{R}^{1 \times D}, \quad (4.2.2)$$

where  $\mathcal{K} \subseteq \mathbb{R}^{1 \times D}$  denotes the features extracted from the Transolver model during training.

### 4.3 Problem setting

Plastic bottle design is a critical problem in consumer packaging, where the competing objectives of material reduction and structural performance must be simultaneously satisfied. While minimizing material usage helps reduce environmental costs of using plastic, bottle designs must maintain sufficient strength to withstand stresses experienced in their manufacturing process and transportation. A key requirement for packaging bottles is their ability to resist excessive deformation during top loading which occurs

in storage. Compressive forces exerted during such top loading conditions can induce structure collapse or buckling in bottle. Thus, careful geometry optimization is required to develop reliable bottle designs while minimizing plastic usage.

Traditional bottle design and evaluation follows an iterative process involving computer-aided design (CAD) and finite element analyses (FEA) where candidate designs are generated using CAD, then simulated with FEA and refined until the design objectives are met. This process is computationally expensive, especially when the design space is large since each design evaluation requires full FEA simulation before refinement. Neural surrogates offer a compelling alternative in such scenario since they can generate simulation outputs for multiple geometries almost instantaneously once the surrogate has been trained on historic data. Such an approach can help minimize product development time significantly, especially during concept exploration stage where large design spaces need to be explored rapidly with reasonable accuracy.

### 4.3.1 Geometry selection

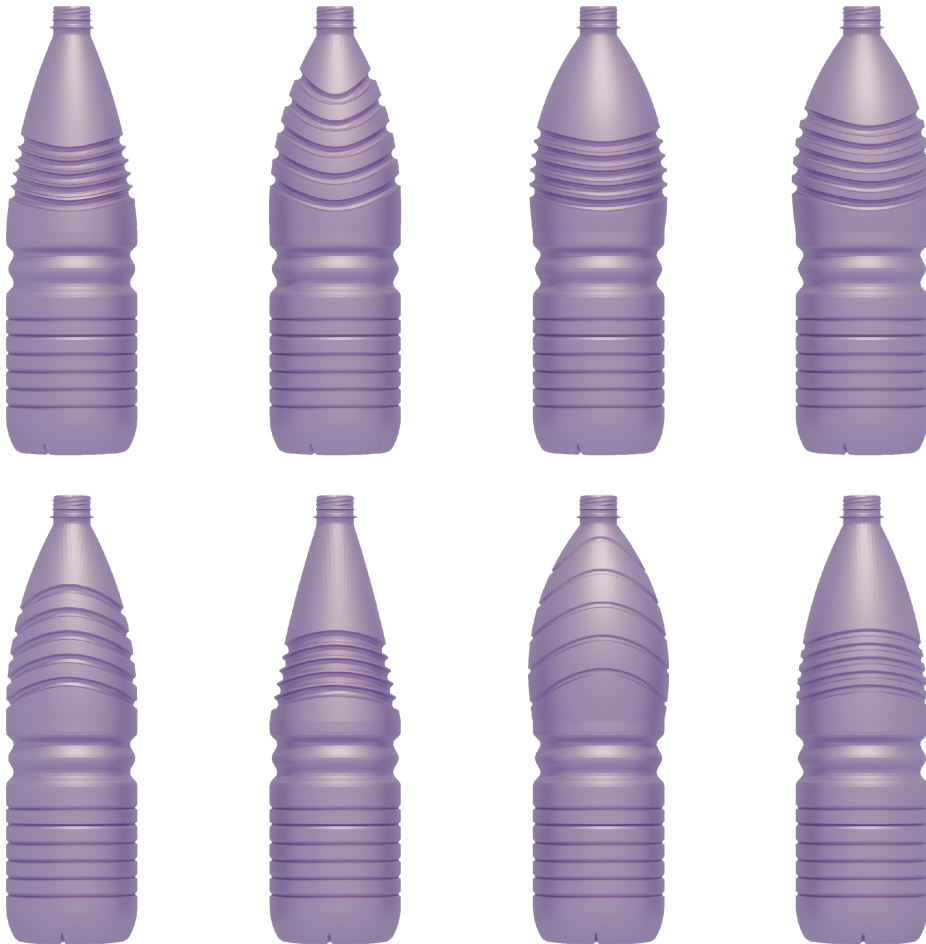
To generate new bottle geometries for our analysis, we employed a parametric design approach that allows systematic variation of key geometric features. Two families of designs were considered, distinguished by the number of parameters varied: two-parameter and four-parameter designs. In the two-parameter family, the top section radius ( $r_{\text{top}}$ ) and rib spacing ( $d_{\text{rib}}$ ) in the upper section were varied to create distinct shapes within a Full Factorial design space  $\mathcal{D}$

$$\mathcal{D} := [20, 35] \times [10, 25] \subset \mathbb{R}^2, \quad (r_{\text{top}}, d_{\text{rib}}) \in \mathcal{D}. \quad (4.3.1)$$

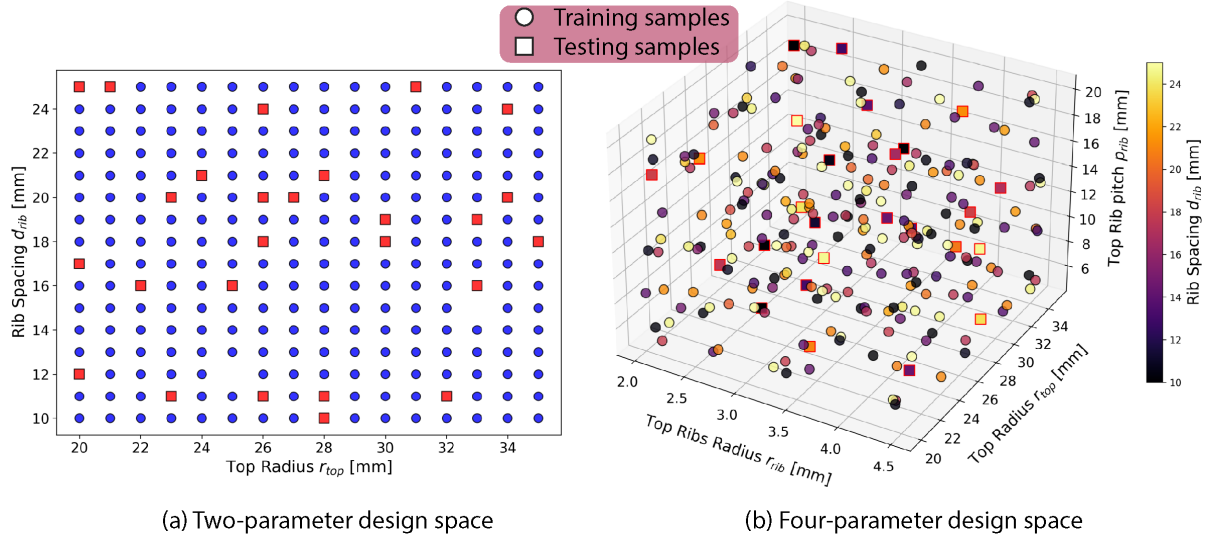
The four-parameter design family adds more design variability by including two additional design parameters: rib radius ( $r_{\text{rib}}$ ) and rib pitch ( $p_{\text{rib}}$ ) such that

$$\mathcal{D} := [2, 4.5] \times [20, 35] \times [5, 20] \times [10, 25] \subset \mathbb{R}^4, \quad (r_{\text{rib}}, r_{\text{top}}, p_{\text{rib}}, d_{\text{rib}}) \in \mathcal{D} \quad (4.3.2)$$

The ranges of these parameters were selected to ensure adequate coverage of design space  $\mathcal{D}$ , and sample selection was performed using an Adaptive Design of Experiment strategy [134] from the Process Composer App in Dassault Systèmes’ **3DEXPERIENCE** platform. This ensures that the chosen designs efficiently span the design space and include sufficient variability for training a useful neural surrogate. For each case, a total of 254 unique designs are generated for FEA analysis. In figure 4.1 we show representative samples from the two bottle families used in this study while figure 4.2 shows the location of chosen samples in the design space for the two bottle families. Note that these design parameters are only used for data generation and are not a part of our neural surrogate framework.



**Figure 4.1:** *Representative samples of bottle geometries generated for this study. Top row represents two-parameter bottle samples while the bottom row shows four-parameter bottle samples. Four-parameter bottle family includes higher geometric variability due to a larger design space  $\mathcal{D}$  and poses a more challenging setup for training a neural surrogate.*



**Figure 4.2:** Visualization of design space  $\mathcal{D}$  for the two bottle families. Here, circles denote the training samples while squares denote the test samples chosen in this study. (a) The two parameter design uses a uniform sampling strategy to cover the design space  $\mathcal{D} \subset \mathbb{R}^2$ . The missing samples in the design matrix are a result of bottle geometries that failed to converge in simulation. (b) The four-parameter design uses Adaptive sampling to efficiently cover the design space  $\mathcal{D} \subset \mathbb{R}^4$  while minimizing computational cost. Note that the four-parameter design space results in more pockets of under-sampled regions.

### 4.3.2 Simulation details

We consider a displacement-driven problem to simulate top load condition for our bottle designs. In the context of bottle buckling analysis, prescribing displacement ensures that the simulation always produces a complete load–displacement curve. This includes capturing the peak reaction force, which represents maximum load-bearing capacity, as well as the potential subsequent softening phase that indicates the onset of buckling. To simulate, we first discretize the CAD designs into mesh-based representation within Abaqus, with meshes containing  $\approx 20,000$  nodes and  $\approx 20,000$  elements. The top cap of each bottle is prescribed a fixed downward displacement of 10 mm while the bottom is fully constrained. The bottle is modeled as an elastic-plastic material and the material, boundary condition, and loading conditions are kept same for all designs. Reaction forces were recorded at a reference point coupled to the nodes of the bottle top. The simulation was performed using Abaqus implicit solver [135] which generates a history of nodal displacement field  $(u_x, u_y, u_z)$  and reaction force  $(F_R)$ . The reaction force curve

is analyzed to evaluate the resistance limit under top load and to determine the onset of buckling, which is indicated by the peak and subsequent decline in the reaction force history. The computational time for each simulation is  $\approx 3$  minutes on one CPU. The simulation failed to converge for two cases in the two-parameter design family (see figure 4.2) and therefore, we removed these samples from this study.

## 4.4 Method

In order to create a *geometry-aware* neural surrogate to learn the nodal displacements and time-dependent reaction forces, we propose a synergistic DeepONet-Transolver framework for learning both simulation outputs simultaneously. Before introducing details for our proposed approach, we provide a brief background on Transolver and DeepONet in the following sections.

### 4.4.1 Transolver

Conventional neural surrogates designed for mapping solutions to PDEs on multiple geometries rely on propagation of node coordinates and/or the node connectivity information. This approach enables learning low-level relationships between mesh points but not capturing the correlation between regions of geometry under similar physical states. The objective function for such surrogates is to map mesh-based information to the nodal outputs obtained from solving the PDE. Transolver solves this by shifting perspective: instead of directly working with raw mesh information, it groups them into a smaller set of latent physical states that exhibit similar response. This allows mapping of similar regions in the domain to physically-consistent slices of data aligned with the PDE solution in these regions.

We consider input as a mesh-based geometric representation  $\mathbf{g} \in \mathbb{R}^{N \times C}$ , where  $N$  represents the number of nodal points and  $C$  represents the input feature of each nodal point. These input features may include spatial coordinates, normal vectors for each point,

or zero-level set representations such as SDF. The set of input points can be mapped to  $S$  physically-consistent slices using the following steps:

1. Project  $\mathbf{g}$  to a higher dimension,  $\mathbf{v}$ , using a linear projection such that

$$\phi : \mathbb{R}^{N \times C} \longrightarrow \mathbb{R}^{N \times D}, \quad (4.4.1)$$

where  $\phi$  represents a multi-layer perceptron with activation function and  $\mathbf{x} = \phi(\mathbf{g})$ .

2. Rearrange the projected feature vector  $\mathbf{x}$  into  $S$  slices across  $H$  attention heads

$$\mathbf{x} \xrightarrow{\text{Rearrange}} \mathbf{x}^*, \quad \mathbf{x}^* \in \mathbb{R}^{H \times N \times S}. \quad (4.4.2)$$

3. Attribute weight  $\mathbf{w}$  of each mesh point  $N$  to each slice using softmax operation along slice dimension  $S$

$$\mathbf{w}_{hjs} = \frac{e^{\mathbf{x}_{hjs}^*}}{\sum_{s=1}^S e^{\mathbf{x}_{hjs}^*}}, \quad \mathbf{w}_{hjs} \in \mathbb{R}^{H \times N \times S}. \quad (4.4.3)$$

The token embeddings for dot product attention are obtained for each slice by aggregating across all  $N$  points

$$\mathbf{z} = \frac{\sum_{j=1}^N \mathbf{w}_{hjs} \tilde{\mathbf{x}}^*}{\sum_{j=1}^N \mathbf{w}_{hjs}}, \quad \mathbf{z} \in \mathbb{R}^{H \times S \times D_h}, \quad (4.4.4)$$

where  $\tilde{\mathbf{x}}^* \in \mathbb{R}^{H \times N \times D_h}$  represents linear projection of the mesh representation  $\mathbf{g}$  using another set of MLP,  $\psi : \mathbb{R}^{N \times C} \longrightarrow \mathbb{R}^{N \times (H \times D_h)}$ ,  $D_h$  denotes the width of the last MLP layer. These token embeddings  $\mathbf{z}$  represent the weighted contribution to each slice from nodal points with similar physical features.

Using these physically-consistent tokens, we apply the dot product attention mechanism

to obtain the attention values

$$\mathbf{z}' = \text{Softmax} \left( \frac{\mathbf{q} \cdot \mathbf{k}^T}{\sqrt{D_h}} \right) \mathbf{v}, \quad \mathbf{q}, \mathbf{k}, \mathbf{v} \in \mathbb{R}^{H \times S \times D_h}, \mathbf{z}' \in \mathbb{R}^{H \times S \times D_h}, \quad (4.4.5)$$

where  $\mathbf{q}, \mathbf{k}, \mathbf{v} = \text{Linear}(\mathbf{z})$  represent the linear projection of token embeddings  $\mathbf{z}$  using a linear layer. During training, the token embeddings in each slice are updated to align themselves according to the similarity of output feature values due to the dot product. Points showing similar output values will be aligned to minimize the dot product during back propagation. This is similar to the token alignment scheme of word embeddings in large language models where the positional proximity of words helps to update the tokens during back propagation.

For reconstructing the required output field  $\mathbf{u} \in \mathbb{R}^{N \times C'}$ , de-slicing operation is performed using the slice weights  $\mathbf{w}_{hjk}$  combined with the attention values using summation across slice dimension  $S$  to obtain representation  $\mathbf{y}$  given by

$$\mathbf{y} = \sum_{s=1}^S \mathbf{z}'_{hsd} \mathbf{w}_{hjs}, \quad \mathbf{y} \in \mathbb{R}^{H \times N \times D}. \quad (4.4.6)$$

This is reshaped into feature embedding  $\mathbf{y}^*$  across all mesh points  $N$  through

$$\mathbf{y} \xrightarrow{\text{Rearrange}} \mathbf{y}^*, \quad \mathbf{y}^* \in \mathbb{R}^{N \times D}. \quad (4.4.7)$$

Finally, the output features  $\mathbf{u}$  are obtained following translations through MLPs

$$\mathbf{u} = \phi(\mathbf{y}^*), \quad \mathbf{u} \in \mathbb{R}^{N \times C'}, \quad (4.4.8)$$

where  $C'$  represents the number of output features of the solution.



#### 4.4.2 DeepONet

For Banach spaces  $\mathcal{U}$  and  $\mathcal{S}$  defined as

$$\mathcal{U} = \{\Omega; u : \mathcal{X} \rightarrow \mathbb{R}^{d_u}\}, \quad \mathcal{X} \subseteq \mathbb{R}^{d_x} \quad (4.4.9)$$

$$\mathcal{S} = \{\Omega; s : \mathcal{Y} \rightarrow \mathbb{R}^{d_s}\}, \quad \mathcal{Y} \subseteq \mathbb{R}^{d_y}, \quad (4.4.10)$$

where  $\Omega \subset \mathbb{R}^D$  denotes the open bounded domain,  $\mathcal{U}$  represents the set of admissible input functions and  $\mathcal{S}$  corresponds to the output function space. The neural operator approximation that maps  $\mathcal{U} \rightarrow \mathcal{S}$  is defined as

$$\mathcal{H}_\theta : \mathcal{U} \rightarrow \mathcal{S}, \quad \theta \in \Theta, \quad (4.4.11)$$

where  $\Theta$  denotes the finite dimensional parameter space of the neural surrogate. In practice, the model is trained on a dataset of paired input-output functions  $\mathbf{M} = \{(u^{(i)}, s^{(i)})\}_{i=1}^n$ . The deep operator network (DeepONet) [15], introduced as a neural framework inspired by the universal operator approximation theorem [136], has emerged as a powerful tool in operator learning for PDE tasks. Its design couples two neural networks: a branch network, which encodes the input function  $\mathbf{u}$  sampled at finite sensor locations  $\{x_1, x_2, \dots, x_m\}$  and a trunk network, which processes the spatial-temporal coordinates  $\zeta \in \mathbb{R}^d, \zeta := (x, y, z, t)$  at which the operator is evaluated.

For a given realization of input function  $u_1$ , the DeepONet representation of the operator output at  $\zeta$  is

$$\mathcal{H}_\theta(u_1)(\zeta) = \sum_{i=1}^p b_i \cdot tr_i = \sum_{i=1}^p b_i(u_1(x_1), u_1(x_2), \dots, u_1(x_m)) \cdot tr_i(\zeta), \quad (4.4.12)$$

where  $\{b_i\}_{i=1}^p$  are output embeddings from the branch network and  $\{tr_i\}_{i=1}^p$  are embeddings from the trunk network. The network parameters  $\theta = (\mathbf{W}, \mathbf{b})$  are obtained by minimizing a cost function generally through supervised learning.

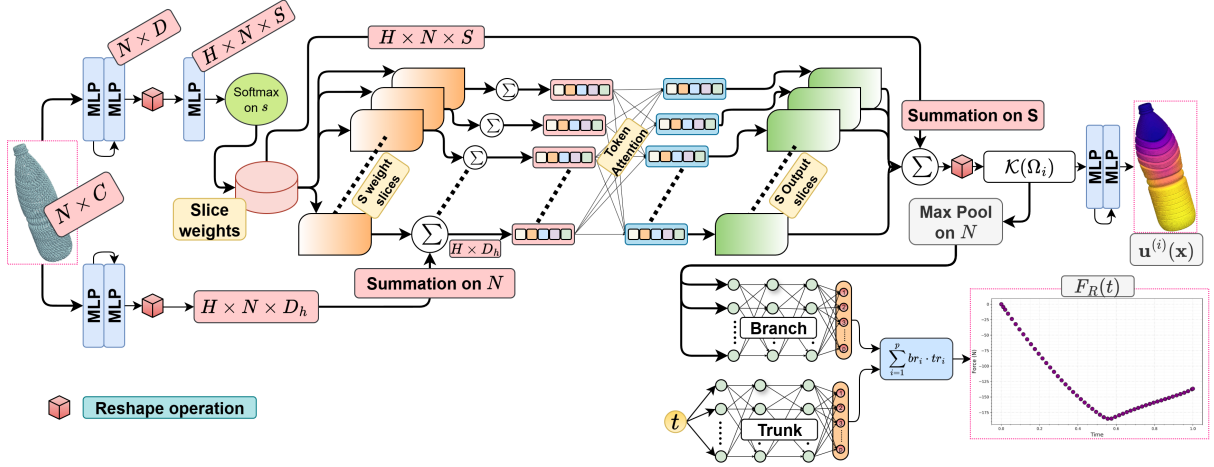
### 4.4.3 Hybrid DeepONet-Transolver framework

The Transolver framework relies on output function values at nodal points to learn and align token embeddings. Surrogate modeling of bottle buckling includes mapping the nodal displacements at the end of simulation time as well as the reaction force experienced at the top of the bottle during this process. While the displacements at the end of simulation can be treated as a static mapping between geometric features to output function and can be mapped using Transolver, the time-dependent reaction force requires additional treatment. In this study, we propose an integrated Transolver and DeepONet approach for simultaneously mapping both the static displacements  $\mathbf{u}(\mathbf{x}, t = t_{end})$  as well as the reaction force  $F_R(t)$ . Figure 4.3 shows a schematic of the integrated framework proposed in this study. Here, the Transolver framework is used for learning the nodal displacement vector for the last simulation time step. The feature embedding  $\mathbf{y}^* \in \mathbb{R}^{N \times D}$  from equation 4.4.7 is obtained as a result of mapping

$$\mathcal{K} : \mathbb{R}^{N \times C} \longrightarrow \mathbb{R}^{N \times D}, \quad \mathbf{y}^* = \mathcal{K}(\Omega_i), \quad (4.4.13)$$

where  $\mathbf{y}^*$  represents a latent representation, and we use this as an input to DeepONet's branch network after applying a max pooling operation across nodal points  $N$ . This helps to create a geometry-specific function representation for individual geometries  $\Omega_i$  during its mapping to displacements  $\mathbf{u}^{(i)}(\mathbf{x})$ . The trunk network uses as input the time steps at which reaction force ( $F_R$ ) is measured during the simulation and is trained with data to learn the map according to equation 4.4.12

$$\mathcal{H}_\theta : Y^* \times T \longrightarrow F, \quad Y^* \subseteq \mathbb{R}^D, \quad T \subseteq \mathbb{R}, \quad F \subseteq \mathbb{R}. \quad (4.4.14)$$



**Figure 4.3:** DeepONet-Transolver framework for simultaneous prediction of nodal displacements  $\mathbf{u}(\mathbf{x}) \in \mathbb{R}^{N \times 3}$  and time-dependent reaction forces  $F_R(t)$  for the top load compression analysis of packaging bottles. Here,  $N$  represents the number of nodal points that defines a point cloud for geometry  $\Omega_i$ , and  $C$  represents the input feature dimension, which comprises of the coordinate locations and surface normals.  $D$  represents the width of the MLP layer,  $H$  represents the number of attention heads for self-attention,  $D_h$  represents the attention head dimension, and  $S$  represents the number of slices into which the inputs features are distributed based on their proximity. The Transolver network predicts the nodal displacement  $\mathbf{u}(\mathbf{x})$  at the last time step  $t = 1$  in the simulation while the integrated DeepONet is used for predicting the reaction force using feature embeddings learnt during Transolver training as input to its Branch network. Both networks are trained concurrently.

## 4.5 Experiments

We evaluate our hybrid DeepONet-Transolver framework with two sets of bottle data: two-parameter design and four-parameter designs. Bottle shapes generated by varying two parameters will have more similarity as compared to four-parameter designs and this enables us to evaluate the performance of our framework for *geometry aware* tasks with different levels of geometric variations. In this section, we present the network architectures and training details along with results of our experiments with the two data sets.

### 4.5.1 Implementation details

The data available for our experiments contains nodal displacements at the end of simulation, reaction force evolution in time, and the mesh-based representation of the bottle geometries. For network training, we generate an input-output dataset  $\mathbf{D} : \{(X_i, Y_i)\}_{i=1}^{N_g}$

with

$$X_i := (\Omega_i, t), Y_i := (\mathbf{u}^{(i)}(\mathbf{x}), R_F^{(i)}(t)),$$

where  $N_g$  denotes the number of geometries. Each geometry input  $\Omega_i := \{\mathbf{x}_j, \mathbf{n}_j\}_{j=1}^N$  in our study is represented by set of  $N \simeq 20,000$  mesh nodes and their corresponding normal vectors ( $\mathbf{n}$ ) extracted using the surface mesh. The output  $\{\mathbf{u}(\mathbf{x}_j)\}_{j=1}^N$  represents the nodal displacements for each  $\Omega_i$  at the end of the simulation step  $t = t_{end}$  while  $F_R(t)$  represents the reaction force experienced at each simulation time step.

## Data pre-processing

We randomly divide the 254 data samples into training (228 samples) and testing (26 samples) using a 90-10% data split. Prior experiments with a 80-20% split resulted in lower overall accuracy ( $\sim 2 - 3\%$  higher error) and we note that a larger training sample size is important to encompass the design space effectively. To improve training stability, both input and output data are normalized using the empirical mean ( $\mu$ ) and standard deviation ( $\sigma$ ) of training data along individual dimensions. For a given variable  $z$ , the normalized form,  $\tilde{z}$  is defined as

$$\tilde{z} = \frac{z - \mu}{\sigma + \epsilon}, \quad \epsilon \ll 1, \quad (4.5.1)$$

where  $\epsilon$  is a small positive constant ( $1e^{-8}$ ) added to denominator for numerical stability. The geometry data  $\Omega_i := \{\mathbf{x}, \mathbf{n}\} \in \mathbb{R}^6$  comprises of the nodal coordinates  $\{x, y, z\}$  and the normal vectors  $(n_x, n_y, n_z)$  concatenated together. The normal vectors are obtained from individual bottle surface meshes using the PyVista [137] library. The output displacement  $\mathbf{u}(\mathbf{x}) := \{u_x, u_y, u_z\}$  contains the displacements along the three axes. The reaction force,  $F_R(t) \in \mathbb{R}$  is evaluated at each time step of simulation, where  $t \in [0, 1]$ .

## Network setup and training

For Transolver, we use an architecture similar to the original work with  $H = 8$  attention heads, MLP layers with 128 hidden features, four physics-based attention blocks with 32 slices, and GELU activation [138]. Layer normalizations, and residual connections are applied as per the original work. We use Pytorch’s Distributed Framework for data parallelism across eight NVIDIA RTX A5500 GPUs to improve the training speed. The Transolver framework is trained to predict the three displacement vectors  $u_x, u_y, u_z$ . For the DeepONet module, we use a branch network with three hidden layers with a width of 128 with GELU activation. Prior to using the Transolver’s encoded output  $\mathbf{y}^*$  with the branch, we also apply a layer normalization for training stability. The trunk network comprises of three hidden layer with width of 128 neurons each and also contains residual connection between layers. The branch and trunk networks generate outputs  $\mathbf{y}_{br} \in \mathbb{R}^{1 \times 128}$  and  $\mathbf{y}_{tr} \in \mathbb{R}^{N_t \times 128}$  respectively, which after summation as per equation 4.4.12 yields the required output reaction outputs  $F_R$  measured at  $N_t$  time steps.

We train the network by minimizing the mean  $L^1$  error between predicted outputs and ground truth values. This choice of loss function was heuristically found to yield better prediction results as compared to standard mean squared loss. The optimization objective is given by

$$\mathcal{L} = \|\hat{u} - u\|_1, \quad \hat{u}, u \in \mathbb{R}^{N \times 1} \quad (4.5.2)$$

where  $\hat{u}$  represents the output from the neural surrogate and  $N$  denotes the number of nodal points defining each geometry. We calculate the loss term for each displacement component and reaction forces separately and obtain the complete loss term given as

$$\mathcal{L}_{total} = \mathcal{L}_{u_x} + \mathcal{L}_{u_y} + \mathcal{L}_{u_z} + \mathcal{L}_{F_R}. \quad (4.5.3)$$

The network parameters for Transolver and DeepONet are optimized simultaneously using Adam optimizer with an initial learning rate of  $1e^{-3}$ . The learning rate is dynamically

adjusted during training using OneCycleLR scheduler [139], which increases the learning rate to a maximum and then gradually decreases it, improving convergence stability and final performance. The network is trained for 10000 epochs with a batch size of one with evaluation step once every 50 epochs. To assess inference quality, we use the relative  $L^2$  norm of error averaged over the number of test samples  $N_{test}$  across  $N$  nodal points in each sample

$$Err = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} \frac{\|\hat{u}^{(i)} - u^{(i)}\|_2}{\|u^{(i)}\|_2}, \quad \hat{u}, u \in \mathbb{R}^{N \times 1}. \quad (4.5.4)$$

## 4.5.2 Main results

In this section, we summarize our experimental findings and provide analysis of our insights from this study.

### Error estimates

In table 4.1, we present a summary of the model’s accuracy across both sets of bottle designs. We include error metrics for the mean error observed on the test data with equation 4.5.4 and also provide the error estimates of the median, lowest and highest error values we observe across individual samples. For the two-parameter bottle designs, the average  $Err$  for the component-wise displacement predictions across all 26 test cases varies between  $\approx 1\% - 13\%$  while the mean error for the reaction force was  $\approx 0.30\%$ . A lower median value of error across all three displacements as compared to mean errors suggest that the model prediction for larger proportion of test cases lies within the  $0.6 - 7.7\%$  error range. Additionally, one example in our test case exhibits significantly higher error for  $u_y$  as noted by the max error value in table 4.1.

The mean error values observed for the four-parameter designs are slightly higher as compared to the two-parameter bottle design, which is expected since the four-parameter bottle shapes has a larger design space,  $\mathcal{D} \in \mathbb{R}^4$ . This means that the design samples generated in this design space have larger separation between them, thus leading to a

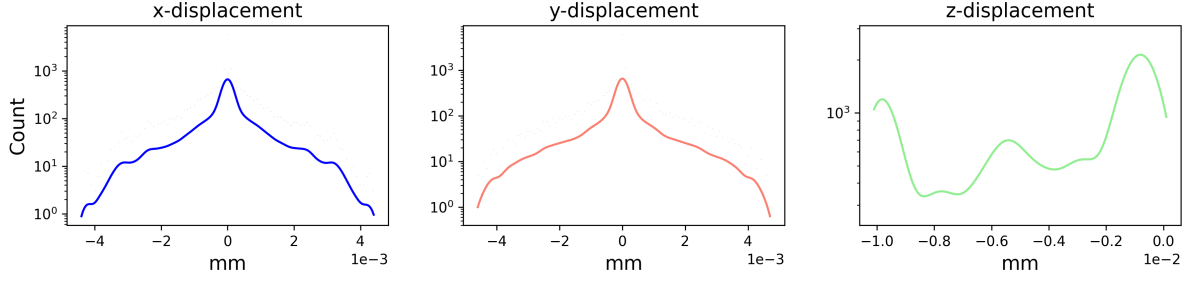
**Table 4.1:** Summary of  $L^2$  relative errors obtained for the two different bottle families. While the overall errors for test cases are reasonable, few test cases show high errors due to these particular samples being under-represented in the training dataset. Note that the best and worst case error estimates represent error across different data samples.

|        | Err: 2-parameter |        |        |        | Err: 4-parameter |        |        |        |
|--------|------------------|--------|--------|--------|------------------|--------|--------|--------|
|        | $u_x$            | $u_y$  | $u_z$  | $F_R$  | $u_x$            | $u_y$  | $u_z$  | $F_R$  |
| Mean   | 0.1049           | 0.1288 | 0.0088 | 0.0030 | 0.1107           | 0.1309 | 0.0245 | 0.0240 |
| Min    | 0.0476           | 0.0424 | 0.0055 | 0.0003 | 0.0336           | 0.0380 | 0.0081 | 0.0044 |
| Max    | 0.2654           | 0.6616 | 0.0267 | 0.0093 | 0.3400           | 0.7090 | 0.0803 | 0.0777 |
| Median | 0.0770           | 0.0616 | 0.0065 | 0.0025 | 0.1013           | 0.1084 | 0.0159 | 0.0166 |

sparser-dataset as compared to the two-parameter samples. Consequently, the entire design space is under-represented in this data. Thus, higher mean and median errors are observed for this data set ranging between  $\approx 2.5 - 13\%$  for the displacements and  $\approx 2.4\%$  for the reaction force. A significantly higher error for  $u_y$  for one particular sample is also observed in this dataset leading to a higher mean error as compared to the median error value. Upon closer inspection, we found that the displacement distribution for this case deviated markedly from the distributions represented in the training set, which can lead to poor generalization. Since the neural surrogate had not been trained on a response pattern of this type, prediction on this particular sample is essentially in an extrapolation regime.

## Point-wise errors for displacement

While  $L^2$  error norms are useful for estimating the overall model effectiveness, an assessment of point-wise errors is important here since the displacement data contains a significant proportion of nodes with zero displacement. In figure 4.4 we show a distribution of displacement data for a representative sample, where a significantly high data accumulation around zero can be seen. This happens due to the fixed boundary condition that exists at the bottom of the bottle, which leads to the ribbed region exhibiting highest displacements during top load. The average  $L^2$  error in such scenarios will be dominated by how well the model can predict those zeros, potentially overshadowing the performance



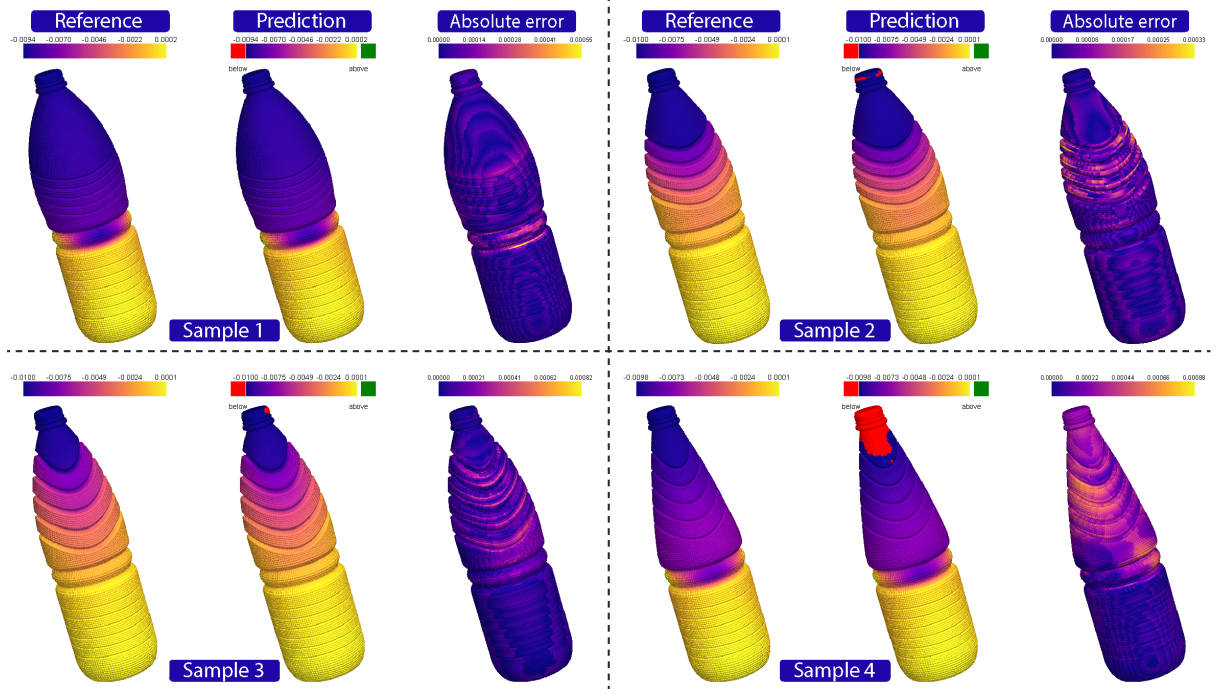
**Figure 4.4:** Plot showing distribution of displacement values for a data sample. A log scale is used for the vertical axis to ensure data is scaled appropriately to show the peak of data frequency around displacement values of 0.0 for  $u_x$  and  $u_y$ . Such unbalanced distribution can bias the average  $L^2$  error estimates.

on the smaller but more important nonzero regions.

Figure 4.5 presents point-wise  $u_z$  displacement predictions for four test samples of the four-parameter bottle family. Across all samples, we observe that the predicted displacement field closely matches the reference solution, capturing both global deformation pattern and localized variations. This indicates that the model can successfully predict the underlying deformation behavior across different geometric shapes with good accuracy. The absolute error plot reveals that the large proportion of the point-wise errors are small (of order  $10^{-4} - 10^{-3}$ ) and are concentrated in specific regions rather than uniformly spread across the domain. The high-error regions occur primarily near the bottle neck and cap region and along transition regions, where corrugated rib patterns meet the smooth surface. Also, in some sample predictions, isolated regions (indicated in red) are found where the model under-predicts relative to the reference solution. The magnitude of absolute error shows that although small localized discrepancies exist, the model predictions are quantitatively accurate overall. Also, the prediction accuracy is consistent across all test samples (except an anomaly with very high error for  $u_y$  displacement), which suggests that the model generalizes well across different geometries. These findings are consistent across the two-parameter bottle designs as well.

To quantify the predictive performance of the proposed learning framework across multiple geometrical configurations, we evaluate the coefficient of determination ( $R^2$ ) between model predictions and reference solutions for the three displacement components.



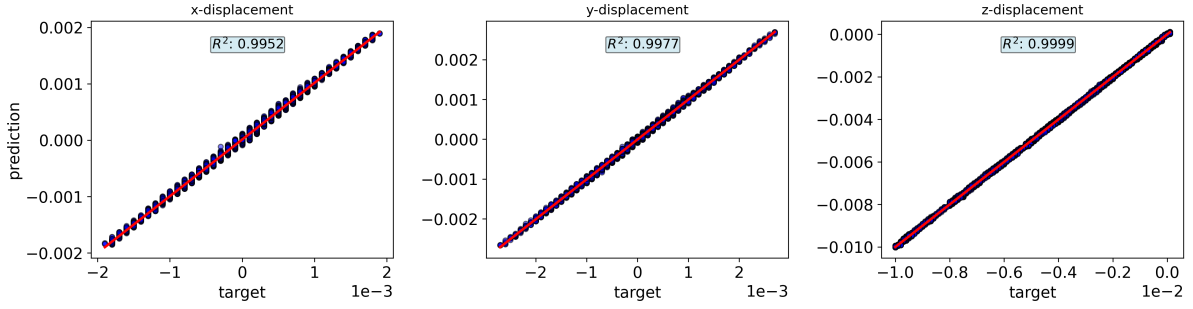


**Figure 4.5:** Point-wise absolute errors in displacement  $u_z$  for four-parameter bottle family test samples. Points in red indicate model prediction that is below the minimum displacement value of reference solution while the points in green indicate predictions that are larger than the maximum value of reference solution.

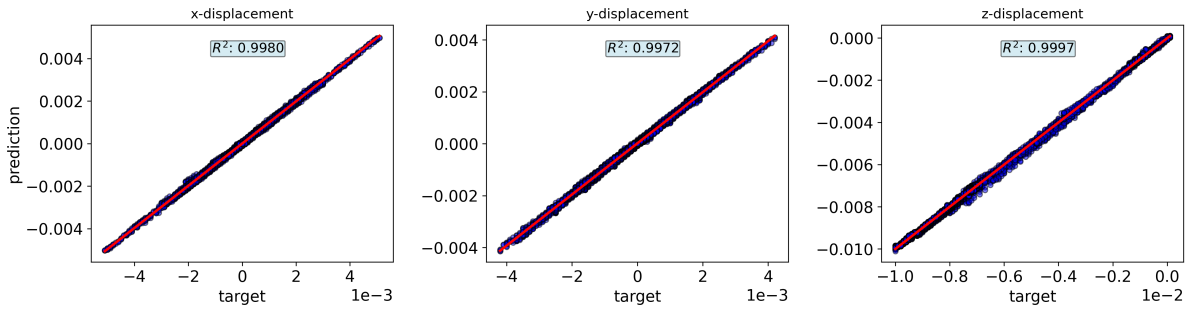
Figure 4.6 shows scatter plots for a test sample from the two classes of bottle designs investigated in this study. The resulting  $R^2$  values for the test samples exceed values of 0.99 in most cases across all displacement components, indicating that the learned surrogate captures a significant part of the variance in target fields.

## Reaction force predictions

Predicting the magnitude of reaction force as it evolves in time is a key element in understanding the buckling characteristic of different bottle designs. In our framework, the integrated DeepONet predicts the reaction force as a function of time using latent features learnt during Transolver training. Figure 4.7 compares the predicted reaction forces against reference data obtained from numerical simulations across different data samples for both sets of data. The plot demonstrates a high degree of accuracy, especially for the samples representing the two-parameter bottle geometries (top row). Note that the profile of the reaction force is relatively simple and monotonic for the two-parameter



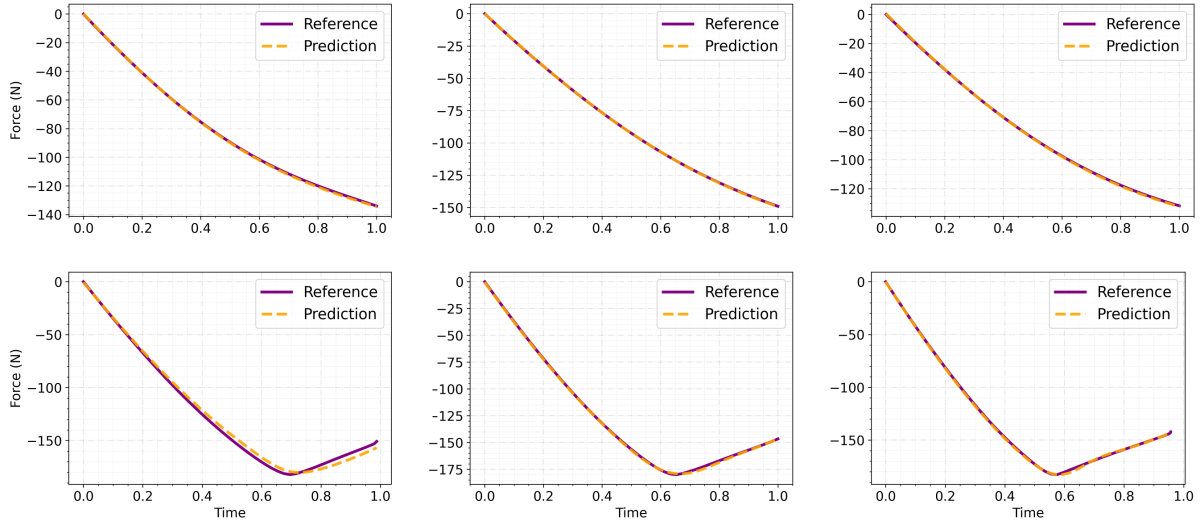
(a) Scatter plot comparing model prediction and reference displacement solutions for a two-parameter geometry sample with  $Err\{u_x, u_y, u_z\} = \{0.0873, 0.0465, 0.0078\}$ .



(b) Scatter plot comparing model prediction and reference displacement solutions for a four-parameter geometry sample with  $Err\{u_x, u_y, u_z\} = \{0.0417, 0.0442, 0.0100\}$

**Figure 4.6:** Scatter plot showing goodness-of-fit of the model for the two bottle design class. The plots show a strong linear correlation between the model’s predicted displacements and the target values for all three spatial components of displacement.

bottle designs because these geometries do not exhibit buckling behavior during simulation. On the other hand, the reaction force profile for the four-parameter geometry samples (bottom row) exhibits complex, nonlinear behavior including a gradient-reversal point which signifies the onset of buckling. Beyond this buckling point, the reaction force experienced starts to reduce. A reasonable match to reference solutions indicated by this plot highlights the model’s ability to capture significant physical events across multiple geometric designs accurately. Here, DeepONet offers a key advantage in time-dependent prediction due to its resolution invariant basis functions generated by the trunk network. This allows us to use as input the time steps from numerical solver and thereby eliminate the need for extrapolation, which would be needed in certain samples where the simulation terminates early ( $t_{end} < 1$ ).



**Figure 4.7:** Comparison of predicted and reference reaction forces versus time for two-parameter (top-row) and four-parameter (bottom-row) bottle designs. The plots show a good agreement between our model’s predictions (orange) and the ground truth data (purple). The model accurately captures the mechanical response across all designs, including the complex nonlinear behavior and buckling events evident in the four-parameter designs.

## 4.6 Limitations and Challenges

In this work, we demonstrated the ability to capture complex nonlinear responses such as buckling events with high accuracy across complex geometric designs using a hybrid DeepONet-Transolver framework. This highlights the promise of geometry-aware neural surrogates as efficient alternatives to computationally intensive finite element simulations in computational mechanics. Nevertheless, we note here some key limitations and challenges observed during this study. One important challenge relates to the sensitivity of surrogate prediction to the choice of training samples, particularly as the design space expands. With a sparse set of training samples available, it becomes crucial to ensure that these samples adequately represent the variability of the design space. Selecting training cases from sparser or more diverse regions of the design space can improve the effectiveness of such surrogates. However, this strategy is less feasible in non-parametric settings, where the design space is unknown or difficult to characterize and often, random sample selection is used in such cases.

Additionally, the two sets of bottle geometry considered in this study are simplified

parameterizations involving two and four design factors. While this problem setting may be sufficient to demonstrate the feasibility and accuracy of the proposed framework, it does not fully capture the complexity of the real-world industrial design problem, where bottle geometries may depend on seven or more parameters. Such higher-dimensional design spaces can introduce richer geometric variations and significantly increase the difficulty of achieving robust generalization. Our preliminary experiments with seven-parameter bottle designs with a smaller sample size revealed that the model’s accuracy decreases notably under such conditions. Hence, extending this methodology to handle higher-dimensional parametric representations is necessary, and we continue to explore ways to improve the capability of current framework to more complex designs with smaller sample size to replicate industrial design tasks.

## 4.7 Summary

We proposed a neural surrogate for learning solution maps for *geometry aware* problems in scientific computing. Specifically, we created a hybrid framework using Transolver and DeepONet for predicting static nodal displacements and time-dependent reaction force simultaneously across varying PET bottle designs under top compressive loading. The dataset was constructed from families of bottle designs parameterized by two and four design variables. For each geometry, nodal displacement and reaction forces were computed using Abaqus. This dataset was used for training our framework that relies on Transolver’s physics-based attention mechanism to predict nodal displacements and the DeepONet for predicting time-dependent reaction force experienced at the top of the bottle during compression. We evaluated our framework’s performance using the  $L^2$  relative norm as our error metric calculated independently across each output. During inference, we observe good prediction accuracy across both the two- and four-parameter designs. For the two-parameter bottle design set, we observed a mean  $L^2$  relative error between 1 – 14% for the displacement fields  $(u_x, u_y, u_z)$  and  $\approx 0.4\%$  for the reaction force

( $F_R$ ). For the more challenging four-parameter designs, we observed slightly higher error values, which are still within acceptable range for this application. Point-wise prediction plots further confirmed that the learned surrogate can capture fine-scale variations of the solution fields with good fidelity. The point-wise absolute error value plot reveals that the large proportion of the point-wise errors are small (of order  $10^{-4} - 10^{-3}$ ) and higher errors are concentrated in specific regions rather than uniformly spread across the domain. These high-error regions occur around the bottle neck and ribbed region of the geometry where strong nonlinear deformation exists during bottle buckling. We also evaluated the goodness-of-fit of our model by estimating the coefficient of determination  $R^2$  between our model’s displacement prediction and reference solution and for most test cases, and we found high  $R^2$  values of 0.98 and higher across all displacement fields. The accuracy of reaction force across the two- and four-parameter design sets was 0.39% and  $\approx 3\%$ , which signifies that the integrated DeepONet module generates a reasonable approximation for time-dependent outputs using latent embeddings from Transolver. This approach extends the Transolver model’s applicability to other problems requiring multi-task solutions.

Overall, this work establishes a promising direction for using deep learning models as efficient surrogates for multi-geometry problems in computational mechanics, enabling rapid evaluation of candidate designs without repeated high-fidelity simulations. Our future endeavors are focused on extending the use of this framework for problems with larger design space and improve computational efficiency.

## PART II

# CHAPTER 5

## Introduction to LLM-assisted Multi-Agent Frameworks for Engineering Designs

The engineering design process is a complex, multi-faceted endeavor that extends beyond the execution of numerical simulations. It encompasses human-centric and collaborative activities, involving the synthesis of domain knowledge, the management of competing requirements, iterative refinement, and intricate trade-off analyses. This process is often resource-intensive, relying on the collective expertise of specialists from various disciplines to navigate the uncertain design space in search of a solution. The inherent complexity and reliance on tacit knowledge [140] present significant challenges to efficiency and innovation, creating bottlenecks that can slow the pace of product development.

The recent rise of Large Language Models (LLMs) has introduced a paradigm shift in the role of artificial intelligence in engineering design [141, 142, 143, 144, 145]. These models can fundamentally transform knowledge-intensive workflows by acting as powerful assistants to augment human intellect and automate labor-intensive tasks. The application of LLMs to the structured, yet creative, domain of engineering design offers a new research opportunity to streamline engineering processes, improve the access to sophisticated

engineering tools beyond specialists, and ultimately, accelerate product development cycle.

Our initial exploration into this domain begins with the idea of using a single LLM as a high-level orchestrator for the scientific machine learning workflow. The resulting framework, MyCrunchGPT [146], positioned the LLM as a central conductor, translating high-level user prompts into a sequence of technical operations, including problem formulation, code generation for simulations, data pre-processing, and post-analysis. This work demonstrated the feasibility of leveraging LLMs to lower the entry barrier for complex computational tools, enabling users to perform tasks like airfoil optimization through a more structured and conversational interface. However, this model of a single AI assistant interacting with a human user, while powerful, does not fully capture the collaborative, multi-role nature of a real-world engineering team.

To address this limitation, we focus on a more representative paradigm: the multi-agent system [147]. We propose that the complex engineering design process is better simulated by a team of specialized AI agents, each empowered with a specific role and expertise, that collaborate to achieve a common design goal [148]. This approach mirrors the structure of human engineering organizations and allows for a more robust, scalable, and nuanced automation of the design workflow. The core of our contribution lies in the development, refinement, and demonstration of a knowledge-guided, multi-agent framework to navigate the engineering design process. We first introduce a foundational framework that formalizes the design process through structured, iterative design-and-review loops executed by specialized AI agents [149]. This system comprises key roles such as a Design Engineer, responsible for generating design candidates, and a Systems Engineer, tasked with evaluating those candidates against established requirements. An important component of this work is the explicit grounding of these agents in verifiable domain knowledge [150, 151]. We utilize a Graph Ontologist agent [152, 153] that leverages an LLM to parse technical literature and construct domain-specific knowledge graphs [154]. These knowledge graphs serve as a structured, external memory for the other agents, providing them with the specialized expertise needed to make informed decisions



to replicate the behavior observed in real engineering teams.

Building on this collaborative structure, we then extend the framework to address one of the most critical phases of design: the conceptual design stage, often characterized by a vast parameter space and significant uncertainty. We introduce a human-in-the-loop paradigm where a human Manager oversees a team of agents operating within a set-based design philosophy [155, 156], which focuses on progressively narrowing a large set of initial candidates rather than iterating on a single point design. A key contribution of this advanced framework is the formal integration of risk management strategy [157]. We task an Analyst Agent with performing a global sensitivity analysis [158] to identify critical design parameters and generate actionable heuristics. Subsequently, the agent team uses quantitative risk metrics, such as the Conditional Value-at-Risk (CVaR) [159, 160], to systematically prune the design set, filtering out candidates with a high probability of failing to meet performance targets. This process automates the laborious initial exploration, culminating in the delivery of a curated, risk-assessed set of promising designs to the human manager for final selection.

In essence, this part of the thesis documents the evolution from a single LLM assistant to a sophisticated, knowledge-driven team of collaborative AI agents capable of navigating the complexities of engineering design. By formalizing agent roles, grounding them in domain-specific knowledge, and embedding principles of risk management and set-based design, we present a novel pathway toward AI-guided autonomous systems that can effectively partner with human engineers. This work aims to enhance the efficiency and robustness of the design process, demonstrating how the reasoning and generative power of modern LLMs can be used to improve engineering processes and workflows. Following this introduction, the subsequent chapters will present the MyCrunchGPT concept, the knowledge-guided multi-agent framework, and its extension to risk-aware, set-based design, respectively.

# CHAPTER 6

## MyCrunchGPT: A LLM assisted framework for scientific machine learning

### 6.1 Introduction

With the advancements in deep learning and natural language processing techniques in recent times, Large Language Models (LLMs) have witnessed remarkable development and research interest, both from academia and industry alike. The availability of large text corpus in the form of digital content, coupled with modern language processing techniques such as Transformers [161], has improved the efficacy and accuracy of natural language processing tasks such as language translation, text classification and sentence completion. Some state-of-the-art LLMs include BERT [162], T5 [163], RoBERTa [164], XLNet [165] amongst others. These models are typically trained on large text corpus, contain billions of neural network parameters, and can be adapted to meet a user's specific use case using these pre-trained models [166, 167, 168]. A significant and potentially breakthrough advancement was achieved by OpenAI through the development of Generative Pre-trained Transformer (GPT-3), which has shown remarkable performance for a range of natural

language processing tasks ranging from sentence completion to providing specific responses based on instructions provided by the user [169]. Colloquially named as ChatGPT, the GPT language model is one of the largest natural language processing models (with hundreds of billions of parameters) that exists today and is capable of generating very high-quality, human-like responses to user queries on a wide range of topics.

A key ability of ChatGPT model that makes it a significant improvement over existing language models is its ability to learn and comprehend instructions provided by the user and respond appropriately. In other words, a user can define a desired ‘persona’ for ChatGPT, whereby the responses generated by ChatGPT to user queries can be made akin to human-like response. This opens up the possibility of using ChatGPT for process automation in engineering and natural sciences [170, 171, 172, 173, 174, 175, 176, 177]. With its ability to generate suitable code or responses for repetitive, labour-intense tasks, which are often encountered in engineering and natural sciences alike (for instance, writing codes for visualization or creating a patient after-visit summary), there exists an opportunity to use the generative capability of ChatGPT for improving processes with well-defined goals. This idea of using the capability of AI tools to automate decision-making has been explored in connection with expert systems since 1970s. Dendral and Meta-Dendral [178] developed pioneering work that combined the capability of machine learning to identify molecular structures of unknown compounds. This work helped demonstrate the capability of AI-powered expert systems to automate domain-specific complex tasks.

In this present work, we develop a LLM-based assistant, called MyCrunchGPT, to streamline two specific tasks: (1) design and optimization of 2D NACA airfoils, and (2) solving prototype fluid mechanics problem using PINNs [179]. The design process, as we discuss later, involves a series of steps starting from concept generation for detailed design. We focus on the conceptual design stage to demonstrate the application of MyCrunchGPT Design Assistant that provides a natural-language based interactive environment to guide the user through design and the specific optimization process of a 2D NACA airfoil. The

Design Assistant helps automate the specific task to generate new airfoil designs, analyze these new designs using surrogate models, help the user to optimize the airfoil shape and visualize the results when requested by the user. More broadly, we aim to leverage the idea of generative design [180, 181, 182, 183, 184], a process that involves exploration of larger design spaces using machine learning algorithms, to augment a designer’s options during the conceptual design stage. The key idea here is to use DeepOnet [29], the first deep neural operator that can play the role of a surrogate that can generalize to different airfoil geometries. We also demonstrate how physics-informed neural networks (PINNS) [179] can be used in MyCrunchGPT to compute and analyze the results of prototype flows, and how the user by initiating simple prompts can change aspects such as the problem geometry and training hyperparameters so that the underlying code implementation can change accordingly. The same prompting process can be used in solid mechanics and other simulation and analysis tasks in computational science and engineering.

### 6.1.1 Objective and contributions

The primary objective of this work is to set the foundations for an integrated framework that uses LLMs to simplify use of SciML in everyday tasks in computational science and engineering, accelerating engineering design and scientific discovery. While the conceptual framework is general, the currently implemented framework was dictated by the logistic constraints and lack of flexible APIs for the current generation GPT-4. Within this context, the specific contributions of our work are:

- Develop MyCrunchGPT as an assistant to guide users through specialized processes such as engineering design. As an example, we demonstrate 2D NACA airfoil design, optimization, and verification using a standard CFD solver.
- Demonstrate the use of MyCrunchGPT for assisting users in solving classical fluid mechanics problems using PINNs in various geometries.

- Develop a generic framework for integrating MyCrunchGPT with specialized workflow via a custom interactive web-based interface.
- Facilitate an easy human-computer interaction with a web-based framework and generate an end-to-end report of interaction along with an executive summary.

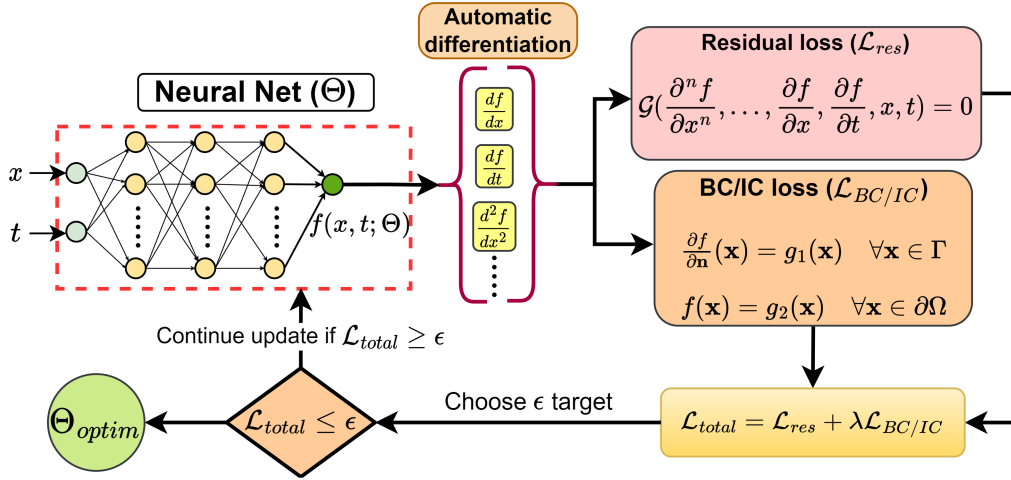
## 6.2 Scientific Machine Learning (SciML)

SciML has emerged as a new form of scientific computing that has the potential to affect greatly all areas of computational science and engineering. Traditional methods that rely on physics-based model require accurate representation of system, which at times is difficult to ascertain for complex systems. As an alternative to physics-based models, data-driven machine learning has become a popular approach in recent times [185, 186, 187, 188]. Deep neural networks, with their universal approximation properties makes it possible to solve partial differential equations (PDEs) or parametrized PDEs while seamlessly injecting any scattered/noisy spatio-temporal data into the solution. This new form of non-sterilized computing has been quickly adopted by the industry for diverse applications [189, 190]. However, many challenges remain associated with neural architecture search, hyperparameter tuning, solution verification and uncertainty quantification. In addition, there is the added cost associated with the problem formulation, data pre-processing, code generation, visualization and analysis, and uncertainty quantification. We propose to overcome all these barriers by building an integrated end-to-end framework that automates all these processes using ChatGPT as the main front-end and in-house methods such as PINNs for PDEs, neural operators such as DeepONet surrogates and for real-time time forecasting [20], but also for other uses, e.g., constructing fast solvers in scientific computing, e.g. see HINTS, [54]. Taken together these developments have the potential to make MyCrunchGPT a friendly and effective research assistant that can continually learn by incorporating new data, new methods, and new LLMs in a seamless fashion. Of particular interest is to produce solutions, which can be verified and validated so

that they satisfy the physical principles, e.g. conservation laws, and agree with available experimental data and existing theories that can be readily provided by ChatGPT.

### 6.2.1 Physics-Informed Neural Networks

Physics-Informed Neural Networks (PINNs) were first introduced in [2], providing a new paradigm for obtaining numerical solutions to both forward and inverse PDEs. PINNs work by representing the PDE solution as a neural network and learning the solution in a semi-supervised learning fashion by data provided by measurements and the PDE evaluated at random points in the spatiotemporal domain. Automatic differentiation is leveraged to encode the PDE residual of the predicted solution into the loss function. A schematic of the structure of a PINN is given in Figure 1.



**Figure 6.1:** Schematic of a vanilla Physics-Informed Neural Network. The unknown solution for a given PDE, denoted by  $f(x, t; \Theta)$ , is obtained as an output of the neural network. Automatic differentiation is used to get the necessary derivatives of the output with respect to its inputs  $(x, t)$ . These derivatives are then used to calculate the residual loss (denoted as  $\mathcal{L}_{res}$  from the original PDE of system and any boundary or initial condition loss (denoted by  $\mathcal{L}_{BC/IC}$ ). The total loss ( $\mathcal{L}_{total}$ ) is obtained as a sum of the two loss terms,  $\mathcal{L}_{res}$  and  $\mathcal{L}_{BC/IC}$ . The term  $\lambda$  represents a weighing factor to balance the loss terms during optimization. Typical loss functions used include the mean squared loss, MSE. The network parameters are updated until the convergence criteria,  $\epsilon$  is reached. Consequently, the required solution function  $f(x, t; \Theta)$  is obtained. See [2] for further details.

Since their inception, many variants of PINNs have been developed, including (but not limited to) conservative, Bayesian [191], fractional [192], parareal [193], variational [194],

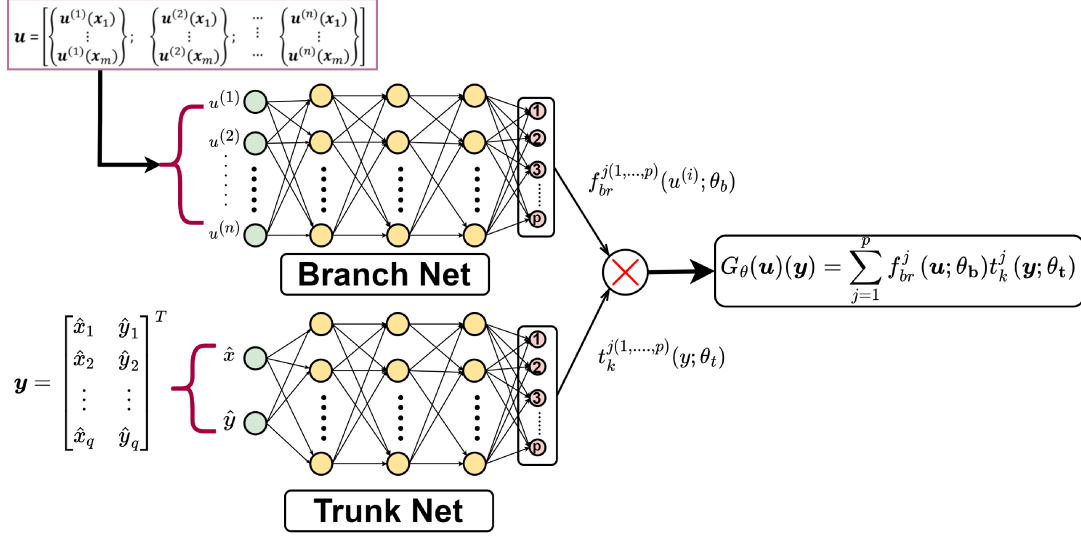
stochastic [195], among others. Since the PINN framework is fairly generic, it lends itself well to problems across a variety of domains. Examples of problems in which PINNs have demonstrated promising results include: Predicting the severity of brain aneurysms using blood flow analysis from MRI videos [196]; Modeling and predicting the behavior of the COVID Omicron variant [197]; Inferring 3D velocity and pressure fields from tomographic background oriented Schlieren videos [198]; quantifying microstructure material properties from ultrasound data [199]; Nonhomogeneous material identification in elasticity imaging [200]; Predicting gas flow dynamics and unknown parameters in diesel engines [201].

### 6.2.2 Deep Operator Network (DeepONet)

A vanilla DeepONet [29] shown in figure 6.2, consists of two deep neural networks, the branch and the trunk net. The branch net encodes the input function  $\mathbf{u}$  at  $m$  fixed locations, also known as sensor locations. The branch network can comprise of multi-layer fully connected network/s or other conventional deep learning architectures such as CNN, RNN, and LSTM. The input function  $\mathbf{u}$  to the branch network is discretized at  $m$  sensor locations such that  $\mathbf{u}^{(i)} = \{u^{(i)}(\mathbf{x}_1), u^{(i)}(\mathbf{x}_2), \dots, u^{(i)}(\mathbf{x}_m)\}$  and  $i \in [1, n]$ , where  $n$  is the number of samples. The coordinate locations,  $\mathbf{y} = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_q\}$  are added as trunk network input to generate the basis function where the solution operator will be evaluated. The solution operator,  $\mathcal{G}_\theta$  is constructed as an element wise product of the output embeddings of the branch and the trunk networks. The optimized network parameters,  $\theta^*$  are obtained by minimizing a loss function ( $\mathcal{L}_1$  or  $\mathcal{L}_2$ ), defined as

$$\begin{aligned}\mathcal{L}_1 &= \frac{1}{n} \sum_{i=1}^n \sum_{j=1}^q |\mathcal{G}(\mathbf{u}^{(i)})(\mathbf{y}_j) - \mathcal{G}_\theta(\mathbf{u}^{(i)})(\mathbf{y}_j)| \\ \mathcal{L}_2 &= \frac{1}{n} \sum_{i=1}^n \sum_{j=1}^q (\mathcal{G}(\mathbf{u}^{(i)})(\mathbf{y}_j) - \mathcal{G}_\theta(\mathbf{u}^{(i)})(\mathbf{y}_j))^2,\end{aligned}\tag{6.2.1}$$

where  $\mathcal{G}_\theta(\mathbf{u}^{(i)})(\mathbf{y}_j)$  is the predicted value obtained from the DeepONet, and  $\mathcal{G}(\mathbf{u}^{(i)})(\mathbf{y}_j)$  is the target value. For more details on DeepONet formulation, readers can refer to [3].



**Figure 6.2:** Vanilla DeepONet architecture showing branch and trunk networks (modified from [3]). The branch network takes functions  $u^{i(1...n)}$  as inputs evaluated at  $m$  sensor locations while the trunk network takes continuous coordinates and parameters,  $\mathbf{y}$  as inputs.

## 6.3 MyCrunchGPT

MyCrunchGPT uses ChatGPT as the front-end and various in-house programs at the back-end that implement PINNs and DeepOnet, supplemented by various modules associated with the geometry, visualization and analysis, etc. These modules and capabilities will be expanded in the near future as GPT-4 APIs become available but for now we present how we use DeepOnet and PINNs in MyCrunchGPT.

### 6.3.1 MyCrunchGPT for Operator-based Design

A typical design process starts with a statement for a product, which is transformed into a set of technical requirements. Engineering design can be described as a systematic process for solving a given problem under the stated constraints. The engineering design process generally involves three stages: conceptual design, embodiment design, and detailed design [202]. The conceptual design stage typically involves idea generation, design space exploration, and preliminary design evaluation. The methodology adopted for conceptual design varies based on the application setting, sometimes even based on user's personal preferences. Over time, designers develop specialized knowledge base and



design methodology based on experience. This presents an opportunity to capture this knowledge-base and experience in an intelligent framework that can be shared and used concurrently by other designers.

Preliminary analysis of design concepts generated during this stage is essential to enable concept selection for further refinement. In engineering design, the analysis tools include CAE, CFD, lumped-system models, and other model-based tools. Often, these tools are computationally expensive and time consuming, especially when multiple designs need to be analyzed. To augment the analysis phase, pre-trained Operator-based Deep Neural networks are useful tools, with their ability to generate analysis results within seconds instead of hours. The DeepONet is generally trained for a specific task, and can help replace or augment the computationally expensive tools that can sometimes inhibit analysis during concept design phase. The ability to analyze multiple designs rapidly also adds value to the design process, since design is typically an iterative process. Hence, an operator-based network such as the DeepONet is an integral part of our framework and acts as a fast surrogate for analyses.

MyCrunchGPT for Deep Operator-based Design presents a structured approach to conceptual design that uses an AI design assistant to guide the user through the conceptual design generation process. The design assistant provides a high level interface to guide the user through a step-by-step approach to accomplish a specific design task. The design assistant uses ChatGPT's ability to generate domain-specific responses to user's queries. With ChatGPT's ability to understand and generate context-based response, the design assistant is tailored to meet specific use cases that are typically encountered during the design process. Additionally, ChatGPT being a LLM, the response generated by the design assistant can also be modified based on user's needs. For instance, the design assistant can enable a novice design engineer to explore specific subjects while guiding through the design process. The assistant's response can then be altered to suit the needs of an expert designer, who may be interested in conducting specific design task and requires less subject-matter exploration. The objective is to eventually create an

AI-assisted expert system to guide the user through the conceptual design phase. Some of the key advantages of such a system include:

- Providing a methodological approach for conceptual design.
- Ability to search and fetch context-based information enabling intelligent workflow creation.
- Tailored to meet specific use cases while having the ability to cater to users of varying technical backgrounds.

### **6.3.2 MyCrunchGPT for PINN-based simulations**

In addition to operator-based design, MyCrunchGPT can assist users in developing customized PINN models and codes for problems in scientific machine learning. The user describes the problem they want to solve using natural language, including relevant details about the PDE, domain, and any other relevant details or files needed to make the problem well-posed (e.g., observed data points, custom geometry files). MyCrunchGPT will formalize the problem the user describes, write and execute code, and provide various postprocessing of the solution if requested. The user can request to visualize different components of the solution, or verify the solution with existing solutions or other numerical codes. The user can request to modify the network architecture and other hyperparameters (optimizer, learning rate, weighting of boundary loss and interior loss, etc.), and MyCrunchGPT will modify and execute the code appropriately.

## **6.4 Exemplar for MyCrunchGPT assisted Operator-based Design framework**

We demonstrate the application of MyCrunchGPT as a design assistant using the 2D NACA airfoil design and optimization problem [4]. The problem statement is described

as follows:

**Problem statement**

Create 2D NACA 4 digit airfoil designs based on maximum camber ( $m$ ) and position of maximum camber ( $p$ ) specified by the user. Generate several new designs as requested by the user. Once generated, analyze the flow field around these new designs. Also, generate an optimized airfoil geometry within the range of parameters  $m$  and  $p$  defined by the user.

This problem was chosen as an example to demonstrate the MyCrunchGPT integrated design process workflow due to availability of existing DeepONet surrogate and CFD simulation data [4]. We acknowledge the need to extend our MyCrunchGPT framework to other design problems and scenarios and consider this as a future work. Here, we describe the design problem setup briefly, by dividing the overall design problem into smaller subsets for clarity.

### 6.4.1 Problem subset 1: Generating new airfoil designs

The surface of the airfoils are generated using the parametric equations with a chord length of one, given as:

$$y_t = \frac{t}{0.2} (a_0\sqrt{x} + a_1x + a_2x^2 + a_3x^3 + a_4x^4) \quad (6.4.1)$$

$$y_c = \begin{cases} \frac{m}{p^2} (2px - x^2) & \text{if } x < p \\ \frac{m}{(1-p)^2} (1 - 2p + 2px - x^2) & \text{if } x > p \end{cases} \quad (6.4.2)$$

$$\theta = \tan^{-1} \left( \frac{dy_c}{dx} \right) \quad (6.4.3)$$

$$x_u = x_c - y_t \sin(\theta), \quad y_u = y_c + y_t \cos(\theta) \quad (6.4.4)$$

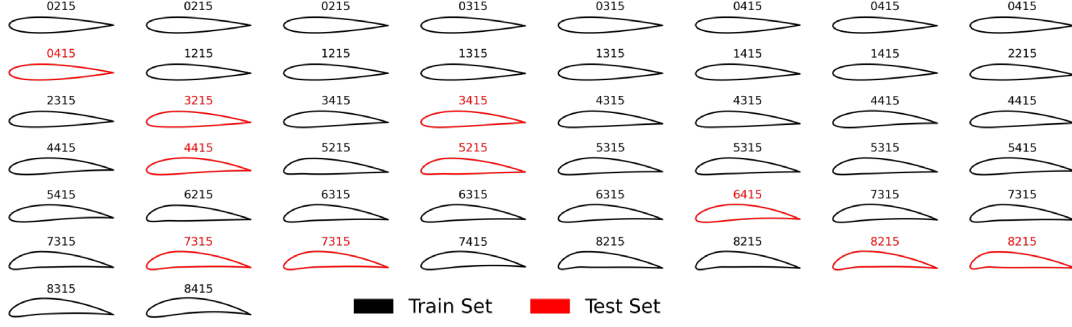
$$x_l = x_c + y_t \sin(\theta), \quad y_l = y_c - y_t \cos(\theta) \quad (6.4.5)$$

where  $a_0 = 0.2969$ ,  $a_1 = -0.1260$ ,  $a_2 = -0.3516$ ,  $a_3 = 0.2843$ ,  $a_4 = -0.1015$ . The 2D

airfoil geometry is therefore defined as a point cloud with coordinate sets  $(x_u, y_u, x_l, y_l)$  parameterized by  $\eta_g = (t, p, m)$ . As a further simplification to reduce likelihood of flow separation and turbulence effects, the parameter space  $\eta_g$  is constrained as follows: the maximum thickness ( $t$ ) is set to a constant value of 0.15, and the domain of the parametric space left by the position of maximum camber ( $p$ ) and maximum camber ( $m$ ) is  $p \times m \in [0.20, 0.50] \times [0.01, 0.09]$ . Here, 100 points are sampled along the x-direction, leading to a set of 100 points  $\{x_{coord}, y_{coord}\}$  that define the coordinates of the airfoil surface. To reduce the dimensionality of the input space for training DeepONet surrogate, Non-Uniform Rational B-Splines (NURBS) with 30 control points are used, hence reducing the input data dimension to 30.

#### 6.4.2 Problem subset 2: Analyzing flow fields around new designs

CFD simulations are widely used for analyzing fluid flow within a domain of interest. Compressible fluid flow simulations using viscous Navier-Stokes equations are often used to study the effect of fluid medium on an object moving through it at supersonic speeds. However, traditional CFD methods using compressible flow solvers are computationally expensive and analyzing several designs simultaneously is a major bottleneck in design exploration phase. As a workaround, fast and accurate surrogate models that can work in tandem with expensive solvers provide a promising alternative, especially in the conceptual design phase where multiple design concepts need to be analyzed. In previous work [4], an operator-based neural network (DeepONet) surrogate model was developed and tested as an alternative to CFD simulation on a set of 2D NACA airfoil designs. The DeepONet surrogate is trained using CFD simulation data generated for forty 2D NACA airfoil designs using Nektar++ [203]. We leverage this pre-trained DeepONet surrogate for analyzing new designs generated through the MyCrunchGPT design assistant. Details for the CFD setup can be found in [4], and figure 6.3 shows the training and testing geometries used for the DeepOnet surrogate training.



**Figure 6.3:** DeepONet surrogate training and testing geometries generated from random sampling from parameter space  $\eta_g$ . Note that the numbers correspond to NACA airfoils and duplicate numbers are due to rounding to the nearest integer for readability. The true NACA parameters are drawn from a uniform distribution and are real-valued. Adapted from [4].

DeepONet surrogate enables one to map from functional space to operator space, which requires domain sub-sampling. In this work, this sub-sampling is accomplished by generating a mesh inside the fluid domain using the free mesh generator tool, Gmsh [204]. We use mesh refinement to reduce the mesh size around the leading and trailing edge of the airfoils. The mesh generation is automated using Gmsh’s Python API, which is integrated with the design parametrization step discussed in section 6.4.1.

### 6.4.3 Problem subset3: Generate optimized airfoil geometry

Shape optimization for geometric shapes is an essential step in design process, where an objective function is minimized/maximized based on a set of constraints. Here, the optimization problem is to maximize the lift-to-drag ratio over a feasible region of parameters, which are  $m$  and  $p$  in our problem. The optimization problem is stated as:

$$\begin{aligned}
 & \underset{m,p}{\text{minimize}} && -LD(m,p) \\
 & \text{subject to} && m_{\min} \leq m \leq m_{\max} \\
 & && p_{\min} \leq p \leq p_{\max}
 \end{aligned} \tag{6.4.6}$$

where  $LD(m,p)$  represents the lift to drag ratio. This lift ( $L$ ) and drag ( $D$ ) are calculated using equation 6.4.7 - 6.4.9. The map created by DeepONet surrogate takes an input

geometry and spatial (x, y) location in the output space and returns the value of the field at that location. This allows estimation of lift and drag through discrete integral of flow fields over the surface of the airfoil

$$\tau_w = \mu(\nabla \mathbf{u} + \nabla \mathbf{u}^\top) \cdot \mathbf{n} \quad (6.4.7)$$

$$D = \int dF_x = \sum p \hat{n}_x dS + \sum \tau_w \hat{t}_x dS \quad (6.4.8)$$

$$L = \int dF_y = \sum p \hat{n}_y dS + \sum \tau_w \hat{t}_y dS. \quad (6.4.9)$$

For calculating gradients of output fields with respect to coordinate locations inside the domain, we use automatic differentiation of computational graph creating during DeepONet evaluation. The analytical derivative  $\frac{dU}{d\hat{n}}$  is given by:

$$\frac{dU}{d\hat{n}} = (v_y - u_x) \sin \theta \cos \theta - v_x \sin^2 \theta + u_y \cos^2 \theta \quad (6.4.10)$$

where the partial derivatives of the velocity field  $\{u_x, u_y, v_x, v_y\}$  are evaluated using automatic differentiation.

For optimization, we use a simulated dual-annealing method inside python library SciPy. Dual-annealing iteratively searches for a potential solution based on an acceptance criteria and draws inspiration from simulated annealing optimization technique [205]. The process for generating an optimized airfoil shape is as follows:

Step 1: Random sampling from parameter space  $p \times m \in [0.20, 0.50] \times [0.01, 0.09]$  using Latin hypercube. These parameters are used as inputs to branch network.

Step 2: Airfoil surface profile generation using equation 6.4.1. Thereafter, NURBS curve fitting to airfoil geometry to define the second input to branch network.

Step 3: Generation of flow field using the DeepONet surrogate using trunk's input coordinates. Calculation of lift and drag from the flow field values.

Step 4: Optimization step, with new parameter selection in next iteration. Repeat until

convergence or user defined step is reached.

#### 6.4.4 Use-case analysis

The use case scenario considered in our MyCrunchGPT Design assistant example is shown in figure 6.4. The user requirements for this design process include:

- Generating a chosen number of new airfoil designs using specified values of max camber ( $m$ ) and its position ( $p$ ).

##### **MyCrunchGPT role**

collects user inputs and calls the appropriate python API to start design generation

- Analyzing the new airfoil designs using the DeepONet surrogate.

##### **MyCrunchGPT role**

initializes the pre-trained DeepONet model, calls necessary APIs to evaluate the new designs through the DeepONet surrogate, and generates the flow fields

- Viewing flow field results.

##### **MyCrunchGPT role**

parses user input appropriately, calls the visualization APIs to plot requested flow field

- Generating an optimized airfoil geometry to minimize lift to drag ratio.

#### MyCrunchGPT role

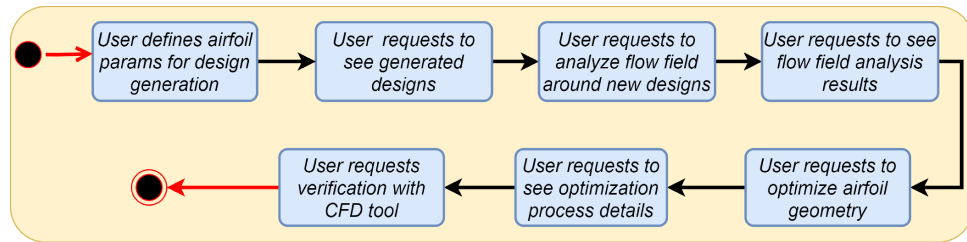
calls the optimization module (using SciPY optimizer). The optimizer module samples the parameter space, creates new airfoil designs based on the sampled parameters, and optimizes the design based on objective function

- Verifying DeepONet results against CFD tool.

#### MyCrunchGPT role

initiates the CFD tool API (Nektar++), requests user for simulation settings, updates CFD simulation script, and initiates the CFD run.

For demonstration, we simplify the use case with a step-by-step process flow diagram (Figure 6.4), although we note that the GPT-based design assistant can learn tasks that are more complex, even cyclical due to the ability of ChatGPT model to understand and learn long command sequences.



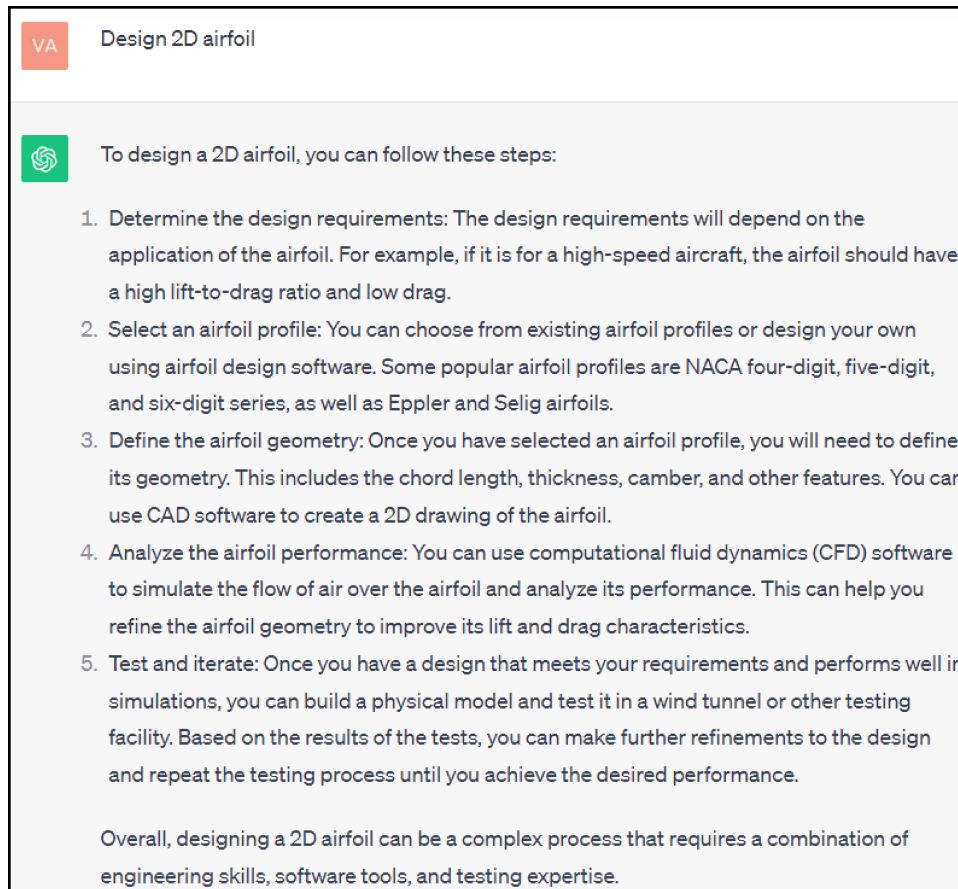
**Figure 6.4:** Use case scenario for 2D airfoil design. The scenario contains expected steps that a user will undertake during a 2D airfoil design process. MyCrunchGPT responds as an assistant during this process to guide the designer while collecting necessary information required for completing user requests. The MyCrunchGPT assistant is instructed to respond to a user entry with an appropriate human-like response.

### 6.4.5 Workflow for DeepONet-based airfoil design

Based on the use case scenario for our MyCrunchGPT, we establish the process workflow required for the design assistant. A necessary step for customizing ChatGPT for the purpose of acting as a design assistant involves creating a sequence of instructions



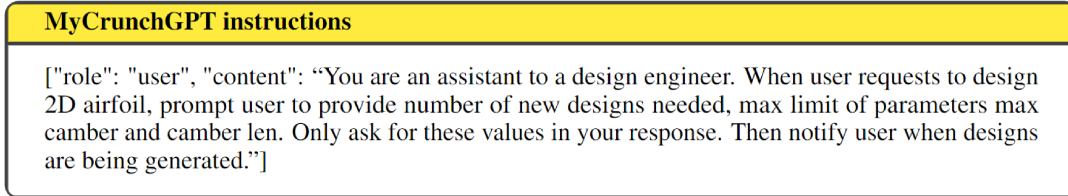
that MyCrunchGPT needs to adhere to when responding to user requests. This is critical since ChatGPT (which forms the backbone for our MyCrunchGPT assistant) is a large language model, and produces generic information. For instance, see figure 6.5 where ChatGPT responds to a user query seeking help with 2D airfoil design with a set of generic guidelines and steps that may be taken. However, this information may not be useful for effective action from a designer.



**Figure 6.5:** Generic responses generated by ChatGPT when the GPT assistant is not primed with instructions for a specific use case. Here, when a user asks ChatGPT to help with 2D airfoil design, a generic response is received. However, for a specific task, such generic responses may not lead to actionable feedback for the designer. Note that this response is generated by the ChatGPT web-based interface (<https://chat.openai.com>).

To receive task-specific responses from ChatGPT, we create a set of instructions to modulate the response based on our use case of the 2D airfoil design. These instructions are appended to the message history using ChatGPT 3.5 API, and set the manner in which ChatGPT should respond to user queries. As an example, we add the following specific

instruction (see Figure 6.6) to ChatGPT API's message history asking the GPT server to request user for specific parameters ( $\text{max\_camber}$ ,  $m$  and position of  $\text{max\_camber}$ ,  $p$ ) when the user requests for generation of 2D airfoil design:



**Figure 6.6:** *Example of instruction used to tailor ChatGPT's response to specific task of designing 2D airfoil based on user query. These instructions are fed to the message history inside ChatGPT thus providing the language model with context for these instructions. This enables us to tune the ChatGPT response to be specific to our tasks.*

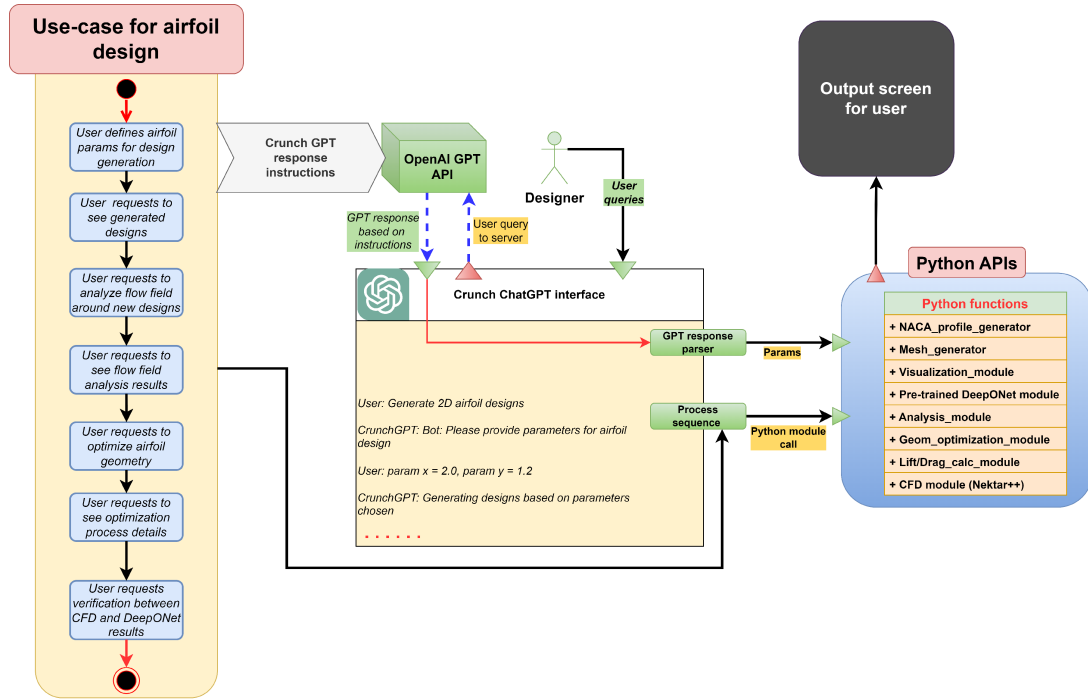
Since ChatGPT is a sequential learning language model, a sequence of such instructions can be passed to ChatGPT API, thereby enabling specific responses to meet the use case requirements. This sequential learning also helps to maintain context for GPT, and this ability is useful for managing complex workflows. Since ChatGPT API currently does not have the ability to perform operations such as read/write or open files on a user's computer, we use pre-defined python functions to accomplish the necessary tasks. The sequence in which these functions are called is determined by the use case scenario shown in figure 6.4. The complete workflow showing MyCrunchGPT for design is shown in figure 6.7. The back-end API for MyCrunchGPT contains a set of python functions required for performing task associated with NACA airfoil design process. The different python APIs used for the design assistant are shown in table 6.1. These APIs are initiated with the required input arguments extracted from user prompts at respective stage of the design process. MyCrunchGPT serves as an assistant for guiding the user through sequence of steps that take the design process towards completion.

Using the workflow as described in figure 6.4, a set of instructions are generated to prime MyCrunchGPT to respond to user queries. A parsing operation parses responses from ChatGPT to extract relevant parameters to execute the next course of action. For instance, when the user enters the parameters  $m$  and  $p$ , the ChatGPT parses this information into a JSON format, which is processed by the parser operation to generate

**Table 6.1:** Name and function of different python APIs used for MyCrunchGPT design assistant framework. Here, each python API executes a pre-defined function from the use-case sequence as defined in figure 6.7. These functions are executed in a sequential order determined by the use-case and input arguments are created by parsing user prompts through MyCrunchGPT interface.

| Python function          | Associated design task                                                                                                                                                                                                                              |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| NACA_profile_generator   | Generates 2-D NACA profile based on user provided max_camber and camber_position.                                                                                                                                                                   |
| Mesh_generator           | Uses gmsh module for generating mesh in a 2D domain with the 2D-NACA profile generated by the NACA_profile_generator module                                                                                                                         |
| Visualization_module     | Used for creating visualizations for flow field                                                                                                                                                                                                     |
| Pre-trained_DeepONet     | A pre-trained DeepONet trained with 2D airfoil velocity, pressure, and density field data generated using CFD. This DeepONet provides a surrogate model for generating flow fields for new NACA geometries created during the optimization process. |
| Geom_optimization_module | Function containing the optimization routine (Simulated Annealing) from python's SciPy library                                                                                                                                                      |
| Lift_drag_module         | Standard equations to calculate lift and drag over airfoil geometry from the field data generated from the DeepONet surrogate model                                                                                                                 |
| CFD_module               | Module for establishing connection to CFD solver (Nektar++), updating solver's setup file and initializing CFD solver.                                                                                                                              |

parameters necessary to call the necessary python operation ('NACA profile generator') in this case. MyCrunchGPT performs the sequence of steps as requested by the user, seeking necessary information and providing appropriate feedback to the user at each stage of the design process.



**Figure 6.7:** Schematic showing workflow for MyCrunchGPT design assistant. Instructions for priming MyCrunchGPT are determined by the sequence of steps outlined in the use-case for airfoil design, as shown on the left side here. These instructions enable receipt of specific response to a user query. The user responses are parsed into required parameters for performing the actions necessary by calling respective python modules.

We demonstrate the response generated by the MyCrunchGPT design assistant at each step in the following section by using screenshots from the user interface.

**Request1 -> User request to design 2D airfoil:** MyCrunchGPT requests the user to provide values for relevant design parameters based on its pre-defined instruction (Figure 6.8).

**Request2 -> User provides requested parameters:** MyCrunchGPT notifies user about the parameter values provided and that the design process is initiating. In the background, the required python function ('NACA mesh generator') is used for generating new design by parsing the parameter values provided by the user; see Figure 6.9.

## Conversation

11:32:16 user: Design 2D airfoil

11:32:18 bot: Sure, how many new designs do you need? Also, please provide the maximum limit of parameter max\_camber and parameter camber\_len.

**Figure 6.8:** [Request 1] MyCrunchGPT's response to user query to design 2D airfoil asking for parameter values  $m$  and  $p$  as well as total number of new designs that are required. Compare how the response has been tailored for our use-case vs a generic response generated by ChatGPT shown in figure 6.5.

## Conversation

11:48:01 user: Design 2D airfoil

11:48:04 bot: Sure, how many new designs do you need? Also, please provide the maximum limit of parameter max\_camber and parameter camber\_len.

11:48:08 user: num\_design = 10, max\_camber = 0.09, camber\_len = 0.5

11:48:19 bot: Thank you for providing the values. Here are the parsed parameter values in JSON format: {"num\_design": 10, "max\_camber": 0.09, "camber\_len": 0.5}. The designs are being generated now. If you want to open the generated designs or show design results, the results will show up in an image viewer pop-up.

11:48:20 system-progress: Generating airfoil designs

11:48:20 system-info: Design generation complete

11:48:20 system-info: Generating mesh for new designs

11:48:25 system-info: Mesh generation completed

Process feedback

Generating airfoil designs

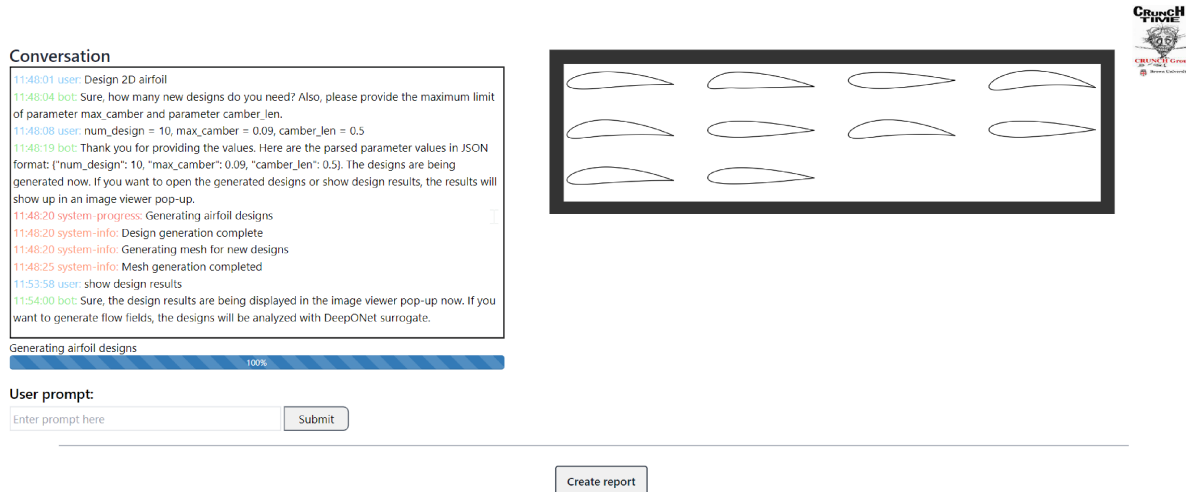
100%

**Figure 6.9:** [Request 2] Parameter values provided by user which are parsed by MyCrunchGPT to provide feedback to user and initiate new airfoil design process by invoking the necessary python function. MyCrunchGPT provides continuous process feedback to the user about progress.

**Request3 -> Display results:** When the user requests MyCrunchGPT to display generated designs, the visualization routine is called in the back-end to display the new airfoil designs generated. The results appear on the webapp's image viewer window; see Figure 6.10.

**Request4 -> Generate flow fields:** When the user requests MyCrunchGPT to analyze flow fields around the new airfoil designs, the designs are analyzed using the surrogate DeepONet in the back-end ([4]). This surrogate generates the flow field around the new airfoil designs by calling the necessary python function (Figure 6.11). The user is notified once the design analysis process is completed.

**Request5 -> Show flow fields:** A user, after looking at the airfoil designs may wish to check the flow field around a particular design. Upon receiving such a request,



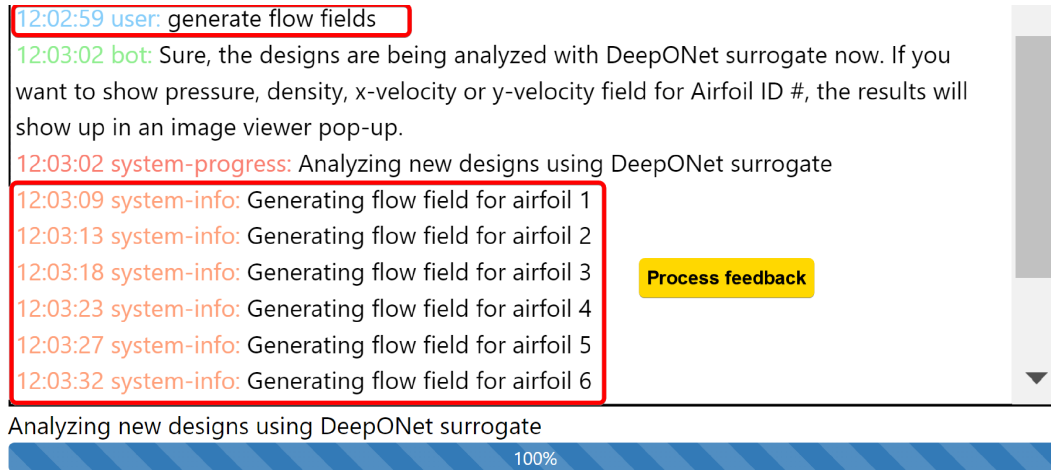
**Figure 6.10: [Request 3]** User requests MyCrunchGPT to show new airfoil designs generated in the design process. Newly generated airfoil designs are displayed in the webapp's image viewer.

MyCrunchGPT invokes a visualization routine in the back-end and the necessary flow field is displayed in the image viewer (figure 6.12).

**Request6 -> Generate optimized airfoil shape:** When user requests MyCrunchGPT to create an optimized airfoil geometry to minimize lift-to-drag ratio, MyCrunchGPT calls the optimization module in back-end that initiates the process. User is notified about the progress through an interface window that shows the evolving airfoil shape with each iteration. The number of optimization iterations required can also be defined by the user (not shown in this example). User is notified of the optimization progress through MyCrunchGPT interface (Figure 6.13).

**Request7 -> Show optimization landscape:** When the user requests MyCrunchGPT to show the optimization landscape for analysis, MyCrunchGPT invokes the visualization module in the back-end to generate necessary plots showing how design is evolved during the optimization process; see Figure 6.14.

**Request8 -> Verify DeepONet results for optimized airfoil against CFD tool:** As a verification step, a user may want to evaluate the optimized geometry with a CFD tool. MyCrunchGPT has an open source CFD tool, Nektar++ integrated in its framework, which can be invoked upon user's request. At this prompt, MyCrunchGPT notifies the user about the CFD tool it can use, the default parameter settings that will

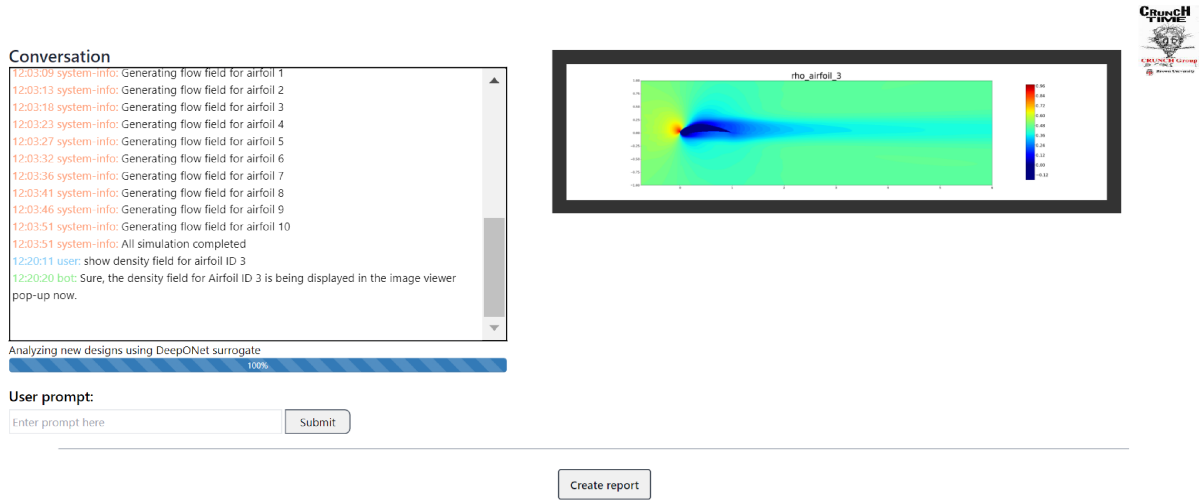


**Figure 6.11: [Request 4]** User requests MyCrunchGPT to analyze flow field around newly designed airfoils. Using this request, MyCrunchGPT calls the necessary python function for analyzing flow fields by calling pre-trained DeepONet. This process takes some time depending on the number of new designs to be evaluated but is significantly less than running a CFD solver. User is provided with continuous feedback while the new designs are being evaluated through the pre-trained DeepONet.

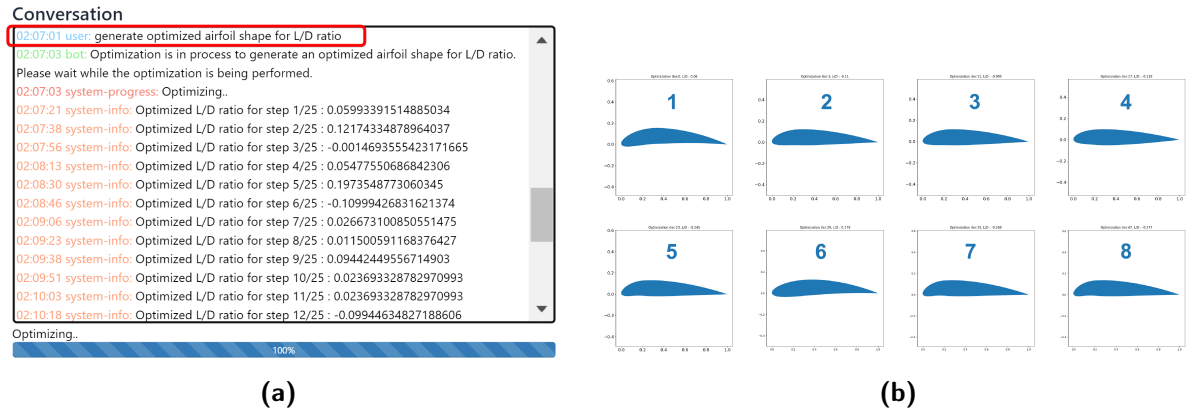
be used for the CFD simulation (Figure 6.15). Additionally, MyCrunchGPT checks with the user to see if any of these settings need to be modified as per user's needs.

Once the user enters the values of the updated parameters for CFD simulation, MyCrunchGPT then parses this information, analyzes the default simulation settings file, identifies the parameters in this file that need to be changed, updates these parameters with user defined values, and then displays the updated code to the user for verification (Figure 6.16a). Once the user confirms these CFD settings, MyCrunchGPT calls the Nektar++ API to start the simulation. The first task performed inside Nektar++ API is to convert the generic fluid domain mesh generated by the 'mesh\_generator' API in python to a mesh that is compatible with Nektar++. Thereafter, MyCrunchGPT calls the Nektar++ API again to initiate the simulation (Figure 6.16b).

**Request9 -> Compare DeepONet predictions with CFD results:** MyCrunchGPT notifies the user once simulation is complete. Thereafter, the user may request MyCrunchGPT to compare the CFD results with DeepONet prediction for the optimized airfoil geometry. At this stage, MyCrunchGPT requests the user to provide the flow field (x-velocity) that needs to be compared, parses the user input, and generates the



**Figure 6.12:** [Request 5] User requests MyCrunchGPT to show density flow field around airfoil 3. The requested density flow field is plotted and displayed.



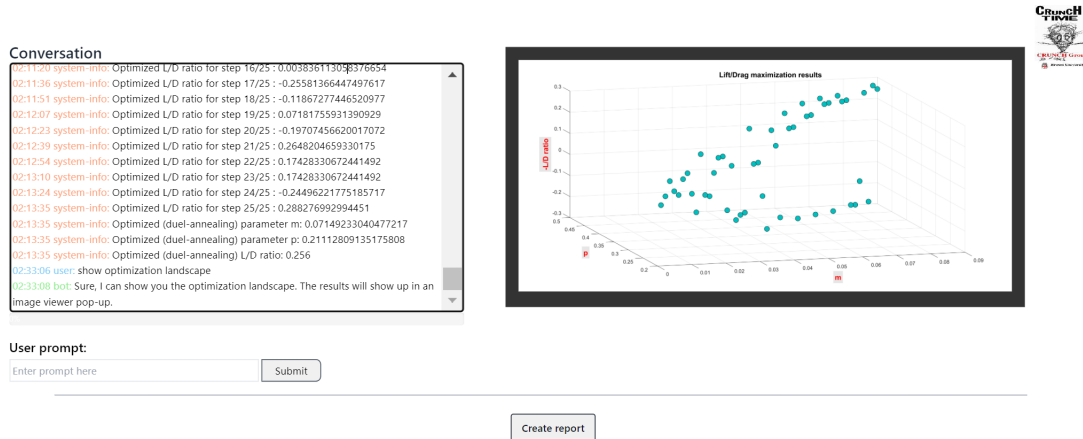
**Figure 6.13:** (a) [Request 6] User requests MyCrunchGPT to initiate optimization process to generate the best profile to maximize lift to drag ratio. MyCrunchGPT calls necessary python modules in the back-end and initiates the optimization process, providing regular process updates through an interface showing how the design is evolving with each iteration. (b) Airfoil geometries being updated during the optimization process displayed to user.

necessary comparison. The user can then request for additional flow field comparison (y-velocity) and MyCrunchGPT generates the appropriate comparison using contextual semantics (figure 6.17). We show the error comparison between CFD and DeepONet in separate plots for reader's clarity (Figure 6.18a - 6.18c). (Figure 6.18d - 6.18f).

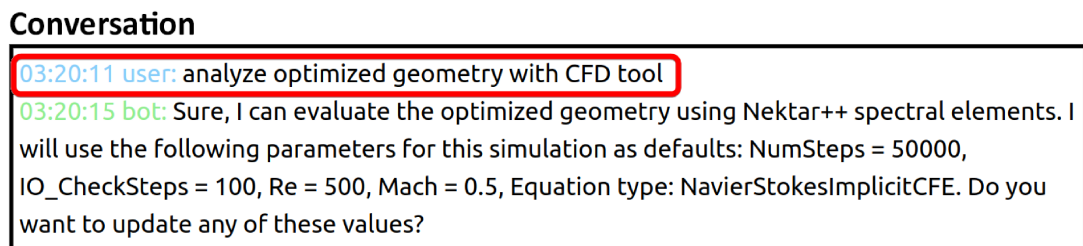
## 6.4.6 MyCrunchGPT assisted result validation and comparison

The extensive language training data used for default ChatGPT model training can also be used for validating and comparing results with existing literature. Here, we



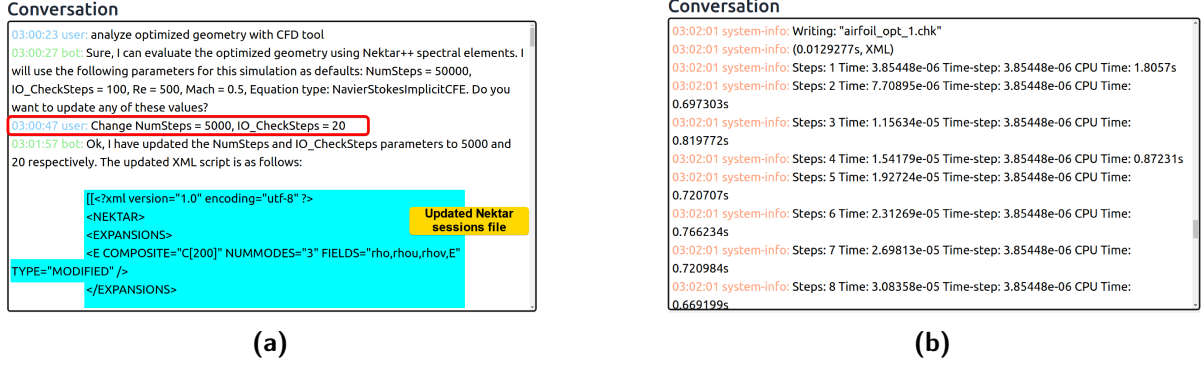


**Figure 6.14:** [Request 7] User requests MyCrunchGPT to show the optimization landscape to analyze design space exploration during optimization. MyCrunchGPT invokes the visualization module to display the required plot to user in the image viewer.



**Figure 6.15:** [Request 8] User request MyCrunchGPT to analyze the optimized airfoil geometry with a CFD tool. MyCrunchGPT then provides the user with the default settings that will be used for the CFD solver, such as number of iteration steps (NumSteps). MyCrunchGPT then waits for the user's response to determine if the CFD solver settings need to be modified or not.

demonstrate a query based result validation process through MyCrunchGPT. We query MyCrunchGPT to identify existing airfoil designs with lift to drag ratio of 0.2, which is similar to was found during the airfoil optimization process. MyCrunchGPT searches for relevant airfoil designs in literature (restricted to the data used for its training) that report the desired lift to drag ratio (see Figure 6.19). The user can further query MyCrunchGPT to identify the necessary details regarding a relevant airfoil to obtain succinct answers. We note that this validation process depends on ChatGPT's ability to search for relevant results from within its training data and may not be accurate for all search cases. In future work, we intend to tune MyCrunchGPT's response to improve its search accuracy and response by creating an instruction set to search appropriate literature databases.



**Figure 6.16:** (a) Updated CFD settings generated by MyCrunchGPT based on changes requested by the user. MyCrunchGPT reads the default CFD settings file provided through GPT instructions, identifies the correct fields (*NumSteps* and *IO\_CheckSteps* in this case) that need to be changed and then updates it with the corresponding user input. Thereafter, MyCrunchGPT displays the updated settings for CFD simulation to the user for user verification (complete MyCrunchGPT modified simulation settings script not shown here for brevity). (b) Once CFD settings are confirmed by user, MyCrunchGPT invokes the Nektar++ API to initiate simulation with constant feedback about simulation progress.

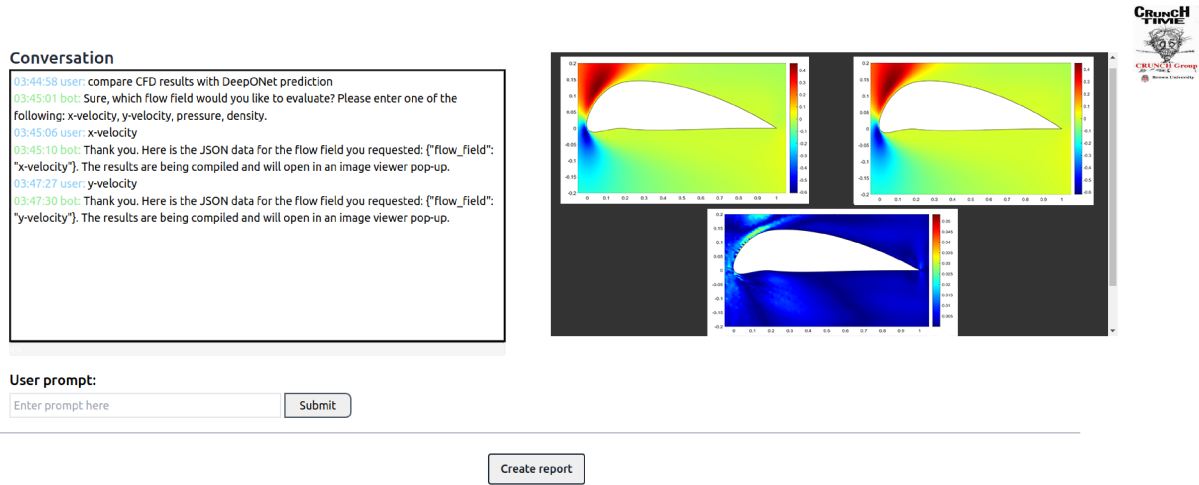
## 6.5 Exemplar for AI-assisted PINN-based analysis

As an example of AI-assisted creation of PINN codes, we will ask MyCrunchGPT to solve the lid-driven cavity problem, a standard benchmark problem in CFD for steady-state incompressible flow. The PDE is defined as

$$\begin{aligned}
 (\mathbf{u} \cdot \nabla) \mathbf{u} &= -\nabla p + \frac{1}{Re} \nabla^2 \mathbf{u} \quad \text{in } \Omega \\
 \nabla \cdot \mathbf{u} &= 0 \quad \text{in } \Omega
 \end{aligned} \tag{6.5.1}$$

where  $\Omega$  is generally taken to be the unit square  $[0, 1] \times [0, 1]$ . Here  $\mathbf{u}$  and  $\mathbf{p}$  are velocity vector and pressure, respectively. The Dirichlet boundary condition on top boundary of the domain is applied by setting the lid velocity to 1 in the x-direction, and no-slip boundary conditions applied to the other walls by setting both components of the velocity to zero.  $Re$  is the Reynolds number, set to 100.

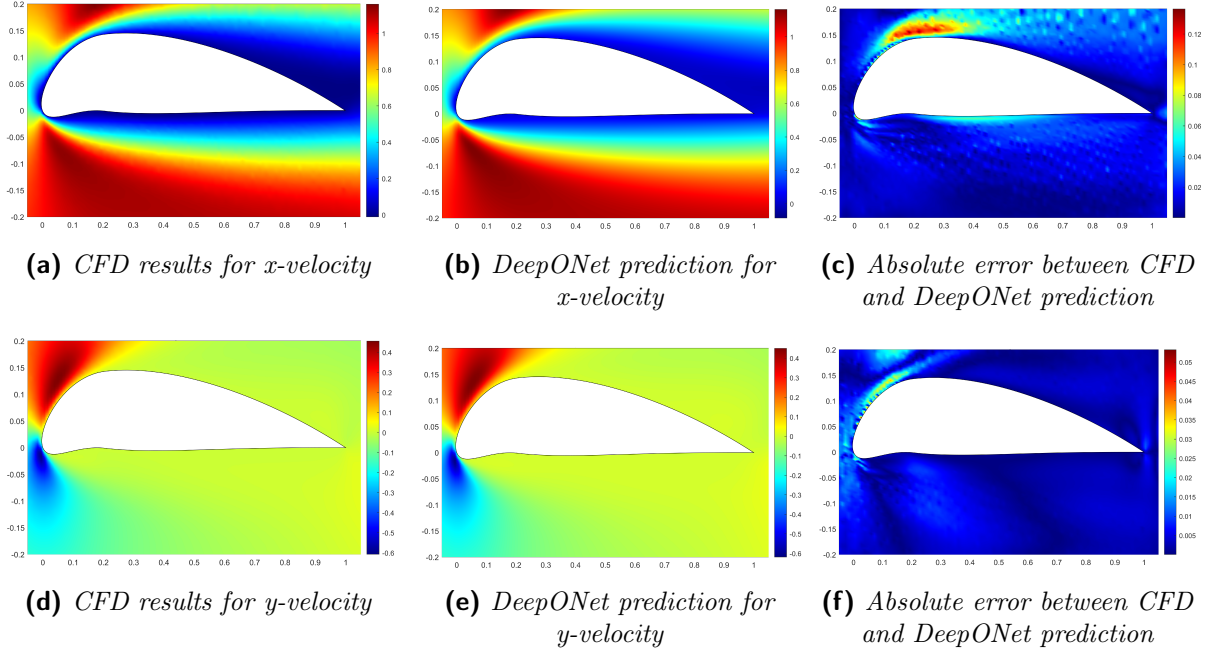
In this work, we present the ChatGPT assisted PINNs for lid driven cavity flow problem on triangular and square domains. We chose two different types of geometry and discretize it with unstructured triangular meshes, as shown in Figure 6.20. The workflow diagram for this problem is given in Figure 6.21. Figures 6.22, 6.23, and 6.24 show the first set of



**Figure 6.17:** *[Request 9] User queries MyCrunchGPT to show flow field comparisons. Here, the  $y$ -velocity comparison between Nektar++ CFD simulation and DeepONet prediction for the optimized geometry is shown in the image viewer.*

chat instructions along with the requested visuals. Figure 6.25 shows the second set of chat instructions, modifying the problem, along with the requested visuals. To verify and validate the results obtained from MyCrunchGPT, we prepared databases for lid driven cavity flow in square domain by digitizing the results from existing literature [5] and integrated in the ChatGPT framework. Therefore, results generated from MyCrunchGPT framework comes with a certificate of fidelity.

Internally, MyCrunchGPT has access to various optimized and thoroughly-tested Python codes we developed for geometry (both 2D and 3D), differentiation, training, verification, and visualization. These codes are developed to be highly modular, so that MyCrunchGPT can compose them easily to achieve the user’s specifications, allowing for flexibility in user request, while maintaining code correctness and largely avoiding hallucinations that LLMs are prone to, especially if asking to code from scratch. We also have a database of example scientific codes and high-accuracy solutions for many benchmark problems that MyCrunchGPT has access to. MyCrunchGPT for PINN simulation is being continuously developed, with new features such as web search, paper Q&A, uncertainty quantification, hyperparameter tuning, among others, being added soon. Multimodality will also be added, with the ability to reason interactively with a given solution visual.

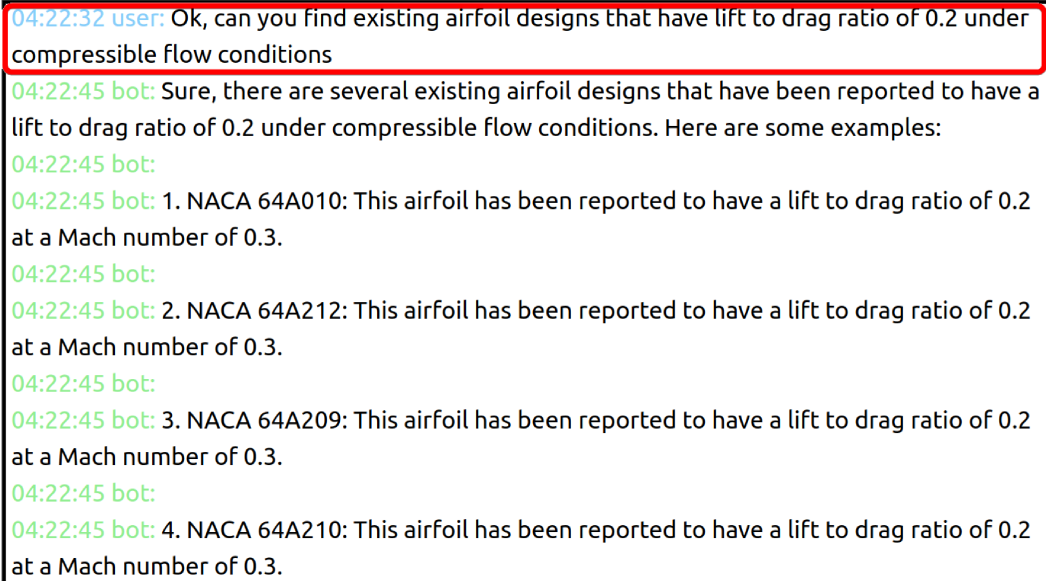


**Figure 6.18:** Comparison for x-velocity between CFD simulation (6.18a) and DeepONet prediction (6.18b). When the user requests to see y-velocity comparison, MyCrunchGPT compiles the necessary visualization APIs to generate the comparison: CFD simulation (6.18d), DeepONet predictions (6.18e), and their absolute error (6.18f). These figures here are intended for reader's clarity and evaluation hence plotted outside the image viewer window.

## 6.6 Limitations

The evolution of ChatGPT as a powerful language model that can generate coherent responses for user inputs provides an excellent opportunity to use these capabilities of Large Language Models in less explored areas such as engineering design and scientific machine learning. While the idea of using ChatGPT as an assistant is appealing, especially with the viewpoint of minimizing redundant tasks, we briefly acknowledge some limitations with our current version of MyCrunchGPT. Using ChatGPT as a design assistant in our example assumes that the design process is well defined that allows us to create a series of instructions for ChatGPT to follow. However, it is well known that engineering processes typically involve cycles of iterations and heuristic exploration of new methods. This presents us with a potential limitation for applications where a sequential series of steps for design process may be ill-defined or unknown. As a strategy, a generic set of instructions that encompass most use-case scenarios expected may be used for providing

## Conversation



04:22:32 user: Ok, can you find existing airfoil designs that have lift to drag ratio of 0.2 under compressible flow conditions

04:22:45 bot: Sure, there are several existing airfoil designs that have been reported to have a lift to drag ratio of 0.2 under compressible flow conditions. Here are some examples:

04:22:45 bot:

04:22:45 bot: 1. NACA 64A010: This airfoil has been reported to have a lift to drag ratio of 0.2 at a Mach number of 0.3.

04:22:45 bot:

04:22:45 bot: 2. NACA 64A212: This airfoil has been reported to have a lift to drag ratio of 0.2 at a Mach number of 0.3.

04:22:45 bot:

04:22:45 bot: 3. NACA 64A209: This airfoil has been reported to have a lift to drag ratio of 0.2 at a Mach number of 0.3.

04:22:45 bot:

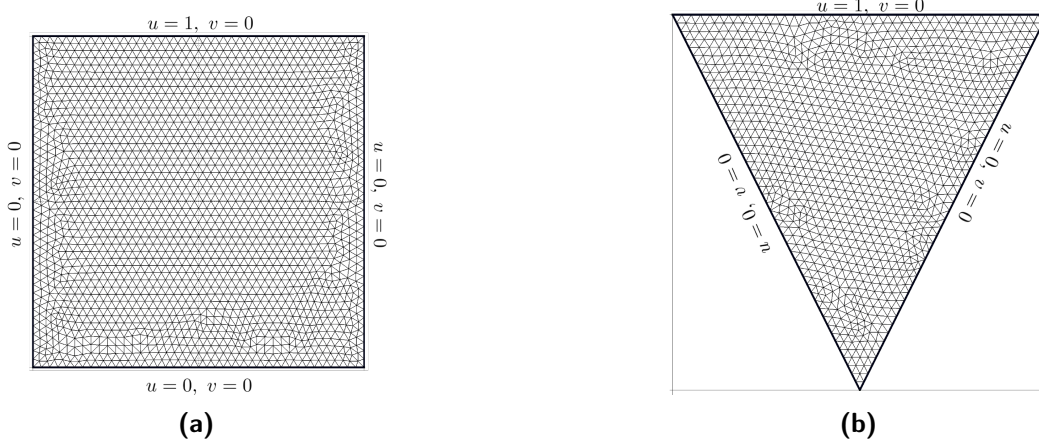
04:22:45 bot: 4. NACA 64A210: This airfoil has been reported to have a lift to drag ratio of 0.2 at a Mach number of 0.3.

**Figure 6.19:** The user requests MyCrunchGPT to compare the specifications of the optimized airfoil geometry with existing airfoil designs found in literature. MyCrunchGPT searches for relevant airfoil designs with a drag to lift ratio of 0.2 and provides a relevant response that can be investigated further by the user.

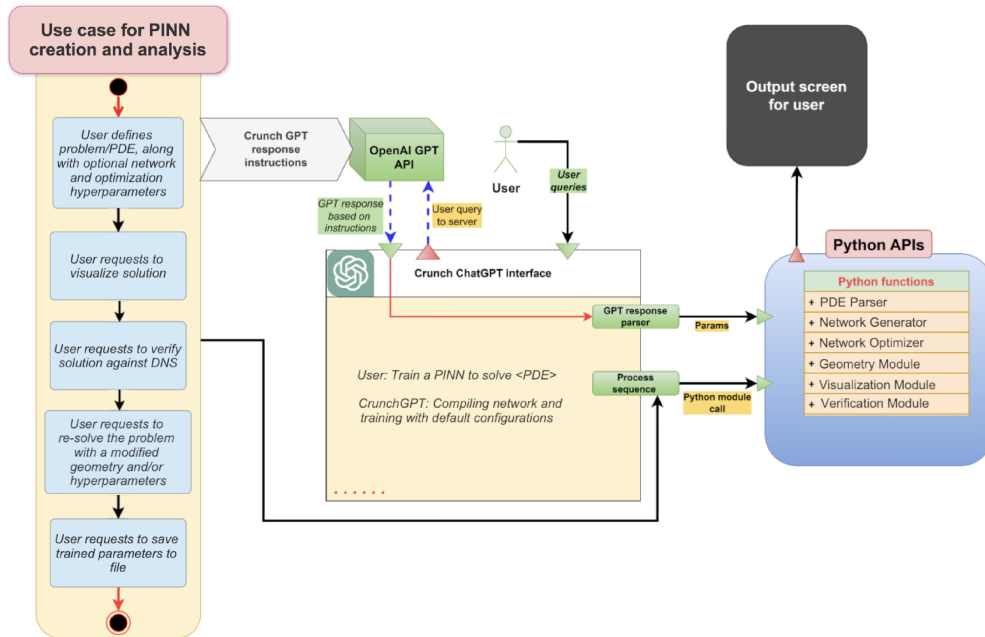
context to MyCrunchGPT assistant. Hallucination is an existing problem with LLMs [206, 207, 208] and since MyCrunchGPT uses ChatGPT 3.5 in back-end, our assistant tool depends on ChatGPT’s capabilities to reduce hallucination. We mitigate this risk by creating instruction sets that are precise yet contain relevant commands to prevent hallucination, as shown in figure 6.6. Moreover, ChatGPT 4 which is an improvement over GPT 3.5, has been found to reduce the risk of hallucination in our recent experiments. Lastly, MyCrunchGPT in this current form cannot address user’s requests when pre-defined functions are not available. We use ChatGPT as a smart-input parser to extract relevant information from user input and then pass these to the requisite function that needs to be called to meet the user’s query.

## 6.7 Summary

In this work, we use two exemplars to demonstrate the use of ChatGPT for scientific machine learning (called MyCrunchGPT) and integration with user-defined workflow as an

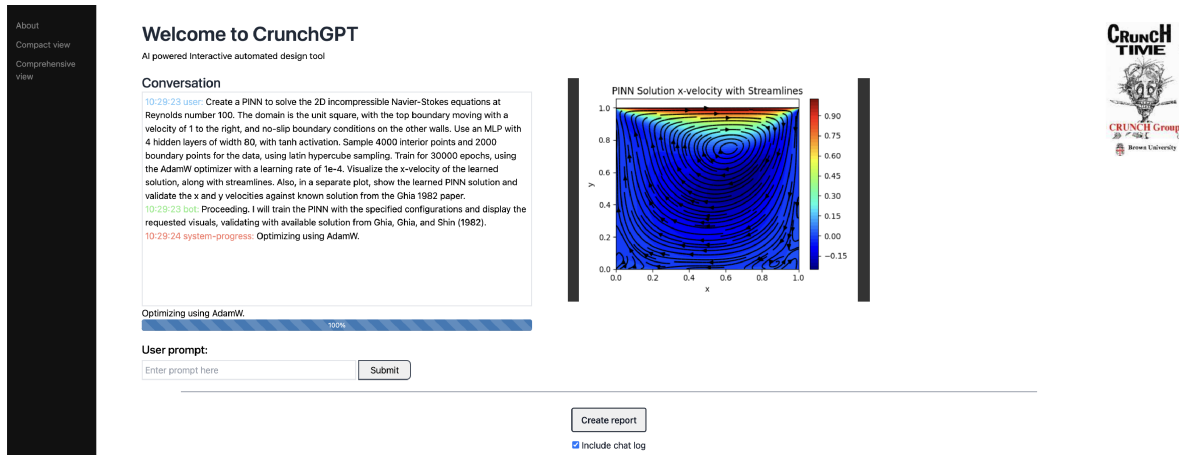


**Figure 6.20:** Discretization of (a) square (b) triangular domain by using unstructured triangular elements used for solving the equation (6.5.1) to validate the velocity fields obtained from PINNs. Equation (6.5.1) is solved by spectral element method provided by Nektar++ frameworks. The boundaries of the domain are marked by respective boundary condition used in formulating the spectral element method. Here  $u$  and  $v$  represent the horizontal and vertical velocity of the fluid, respectively.

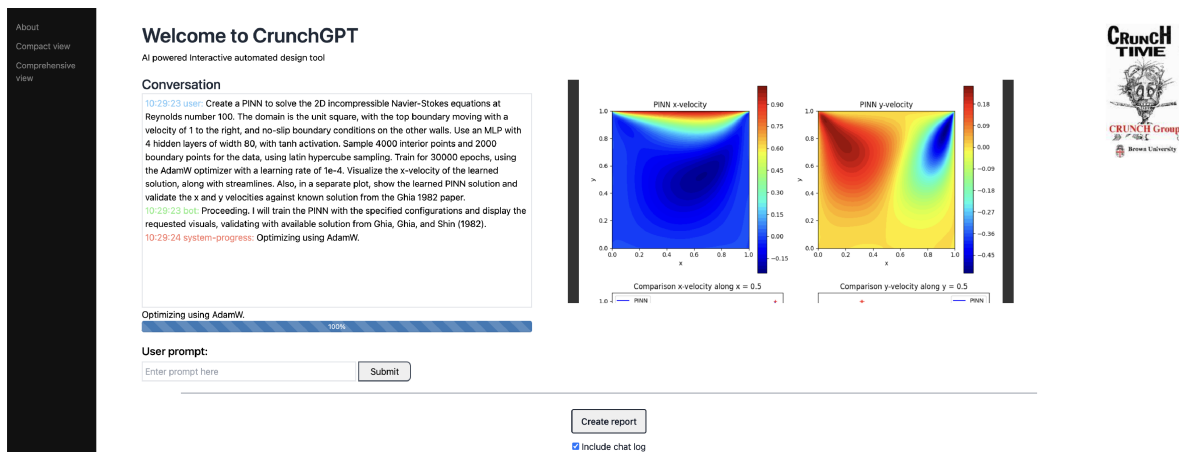


**Figure 6.21:** Workflow diagram for asking MyCrunchGPT to solve a PDE with a PINN. The user defines the PDE and relevant details, and asks to train a PINN for the PDE. The user may provide various requests, such as specifying configurations, visualizing different components of the solution, verifying against known solutions/DNS, or saving/loading trained parameters. The user may ask to modify the problem, referring to the previous instructions without explicitly re-defining the PDE. GPT is used to parse the instructions into components, map those components to the corresponding internal modules, and synthesize the full program by composing the selected modules.



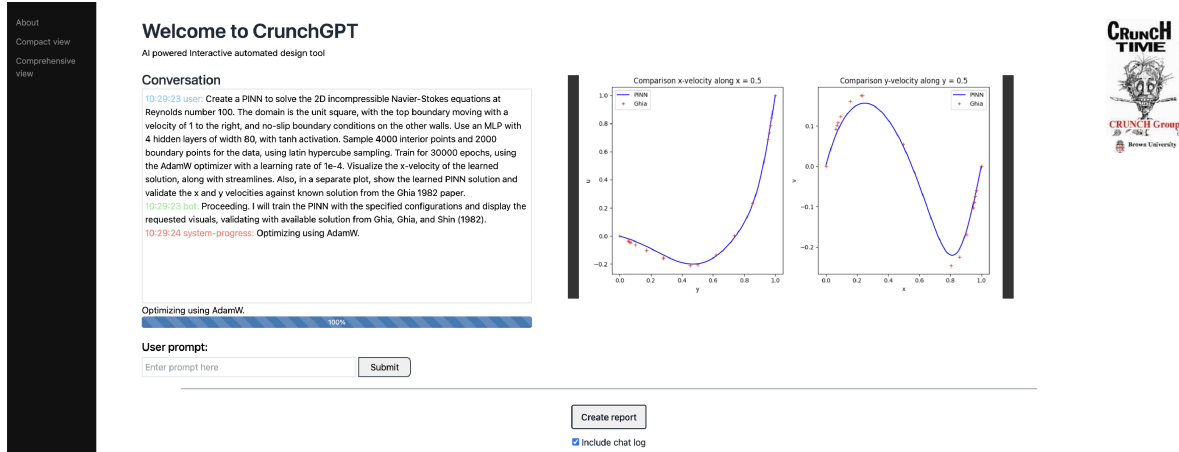


**Figure 6.22:** Asking MyCrunchGPT to use a PINN to solve the lid-driven cavity CFD benchmark problem. The user specifies the PDE to solve, along with the geometry and boundary conditions in natural language, as well as training configurations. The user also requests to visualize the solution in two different ways, one involving validation against a specific known solution. MyCrunchGPT parses the request into the necessary components and synthesizes and executes a program to complete the request.

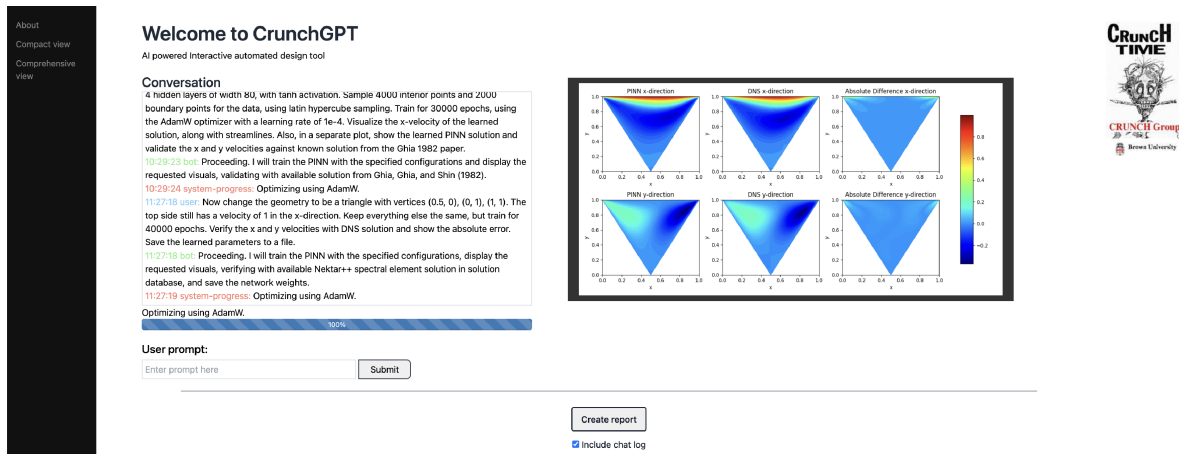


**Figure 6.23:** MyCrunchGPT interface with the first half of the second requested visualization (visualizing the PINN solution).

assistant. As a first example, we tailor ChatGPT's response to assist a user with different steps involved in design and optimization of 2D NACA airfoils. For this, we create a set of instructions for MyCrunchGPT to follow based on user's request and a user defined design workflow. The process starts with the user specifying the need to generate new airfoil designs, which are constrained by the parameters maximum camber ( $m$ ) and position of maximum camber ( $p$ ). MyCrunchGPT then prompts the user to provide necessary parameter values through a conversation window, parses user's requests, and calls necessary python functions in the back-end. Once designs are generated, MyCrunchGPT guides



**Figure 6.24:** *MyCrunchGPT interface with the second half of the second requested visualization (validation with specified known solution from [5]). Currently, MyCrunchGPT’s capability is limited to extracting contextual meanings from references provided in validation stage. A human-in-loop is required for extracting relevant data from images and tables in the specified reference shown here.*



**Figure 6.25:** *Asking MyCrunchGPT to solve a variation of the previous problem. The user requests to change the geometry and training iterations, verify with any DNS solution available, and save the learned parameters, for which MyCrunchGPT modifies and executes the code appropriately.*

the user through the evaluation process through a surrogate DeepONet model. This surrogate model is trained offline with CFD data of airfoil designs, as outlined in [4], and is used to determine the flow field around these new airfoil designs. As the next step, the user requests MyCrunchGPT to optimize the airfoil shape using constraints on parameters ( $m$ ) and ( $p$ ), with the the objective being to maximize the lift to drag ratio. MyCrunchGPT invokes the necessary optimization routine in the back-end and an optimized airfoil geometry is generated. Finally, MyCrunchGPT also assists the user



to compare the DeepONet predicted flow field results with a traditional numerical solver (Nektar++), which is integrated with the MyCrunchGPT workflow. The comparison between DeepONet predictions vs CFD solver prediction for the optimized airfoil geometry show small absolute error. Lastly, MyCrunchGPT provides users with the ability to validate their findings with results available in reported literature. In summary, we demonstrate an instruction-based framework for ChatGPT to invoke appropriate response to guide the user through a design and optimization process.

As a second example, we use MyCrunchGPT to assist in the creation and training of a PINN to solve a specified PDE (the Navier-Stokes equations for the lid-driven cavity benchmark problem), as well as provide visualization and verification of the learned solutions against data from direct numerical simulation. Further requests pertain to validation of the learned solution against available experimental data in the literature. The PINN capabilities of MyCrunchGPT (and the capabilities of MyCrunchGPT in general) are being actively developed to include useful tools such as uncertainty quantification, multimodal solution interaction, hyperparameter tuning, and distributed computation, among others.

The entire process has been embedded in a GUI, based on a webapp. The webapp allows the user to interact with the bot, view the system logs, and observe the figures and outputs of the process. We employed various ways for the user to interact with the GUI, and plan to extend it so more complicated asks are accommodated by the GUI. For example, if the user asks for a table of comparison of certain results. We intend to add more visualization techniques, and improve the way the front-end interacts with the back-end. Last, when the information that appears on the screen is acceptable by the user, they can export it into a summary document. This document also includes a bottom line summary of the process and comparison to benchmarks, done by the bot. We plan to expand this as well, and allow the user to interact with the bot to create more tailored reports, depending on the application.

In future work on MyCrunchGPT, we plan to develop probabilistic set based design

[209, 210, 211, 212, 213, 214], incorporate multiple neural operators such as FNO, WNO, LNO and ViTO [215, 16, 216, 217, 218], and include different versions of PINNs, including domain decomposition and adaptive activation functions [219, 220, 221], that will make CruchGPT more versatile and robust. We will also develop new versions of MyCrunchGPT appropriate for other areas, including systems biology, solid mechanics, materials science, geophysics, and bioinformatics. Lastly, we intend to make this framework more capable of handling new query sequences and benchmark problems by augmenting the code generation scheme through an instruction-driven paradigm.

# CHAPTER 7

## Toward Autonomous Engineering Design: A Knowledge-Guided Multi-Agent Framework

### 7.1 Introduction

The engineering design process serves as the cornerstone of innovation in diverse disciplines. At its core lies an iterative cycle - a structured approach to problem-solving that emphasizes refinement and improvement through repeated cycles of design space exploration, analysis, evaluation, review, and selection. This iterative nature arises due to the complexity of real-world engineering challenges, where final solutions often emerge after multiple rounds of design reviews and requirement clarifications. The process of engineering design, therefore, is not a linear sequence of events but rather an intertwined set of activities, which are often revisited and reordered as the problem understanding deepens. Traditionally, the design process is segmented into three key phrases: conceptual design, embodiment design, and detailed design [222]. The conceptual design phase involves initial exploration of potential solutions, focusing on defining the problem scope, requirement elicitation, generation of several design concepts and their evaluation against requirements.

Often, this phase is characterized by a high degree of uncertainty and repetitions, since the goal is to identify promising directions for further detailed investigation.

In addition to its iterative nature, the design process is also a collaborative endeavor, particularly in the context of complex product development. Interdisciplinary teams, comprising of experts and personnel from diverse specialties are essential for bringing a holistic perspective in this process. Each discipline contributes their unique expertise and knowledge-base, leading to more robust and innovative solutions. This collaborative environment necessitates frequent communication and coordination, often facilitated by specialized design tools and processes. Decision making is an essential step in design and is typically performed by domain experts whose expertise is built with years of practice. This domain knowledge is often tacit, and is difficult to articulate or codify [140]. Such knowledge encompasses not only technical skills but also an understanding of the best practices, potential pitfalls, and subtle nuances of design trade-offs. This accumulated expertise is immensely valuable for ensuring process efficiency, quality, and risk management. However, this reliance on individual expertise presents a significant challenge: the potential loss of critical knowledge during personnel transitions [223]. The departure of experienced engineers can create a void in institutional memory, hindering future design efforts and potentially decline in product quality [224].

The emergence of large language models (LLMs) marks a transformative shift in the traditional engineering design landscape. These models, trained on large collections of text and code datasets, have shown an unprecedented ability to understand, generate, and perform complex tasks when provided with well-designed instructions [225]. The proliferation of LLMs in engineering practice has the potential to reshape the entire design process. LLMs can offer compelling advantages as assistant tools in guiding users through complex, iterative processes that characterize engineering design. In future, one can visualize LLM frameworks capable of generating design concepts, explaining the rationale behind design choices, identifying potential risks and remedies. Such interactive guidance can be particularly valuable for novice engineers, accelerating their learning curve and

fostering a deeper understanding of design principles and engineering tools.

Despite their remarkable capabilities, the application of LLMs for engineering tasks is not without limitations. A particularly significant challenge lies in achieving domain specific knowledge and guidance. Although LLMs excel at general language understanding, they often struggle to provide the deep and specialized knowledge required for complex engineering tasks that frequently require intricate trade-offs and nuanced considerations [226]. Furthermore, the backbone of engineering practice relies on specialized engineering tools—computational fluid dynamics (CFD) solvers, finite element analysis (FEA) software, CAD modeling packages—that have been rigorously developed and validated over many years. Engineers depend on these tools to perform critical calculations, simulations, and analyses. LLMs typically lack explicit knowledge of these tools, making it difficult to accomplish complex engineering tasks solely through LLM-driven interactions without human intervention [227, 228]. The seamless integration of LLMs with existing engineering workflows and tools remains a significant hurdle to overcome [229].

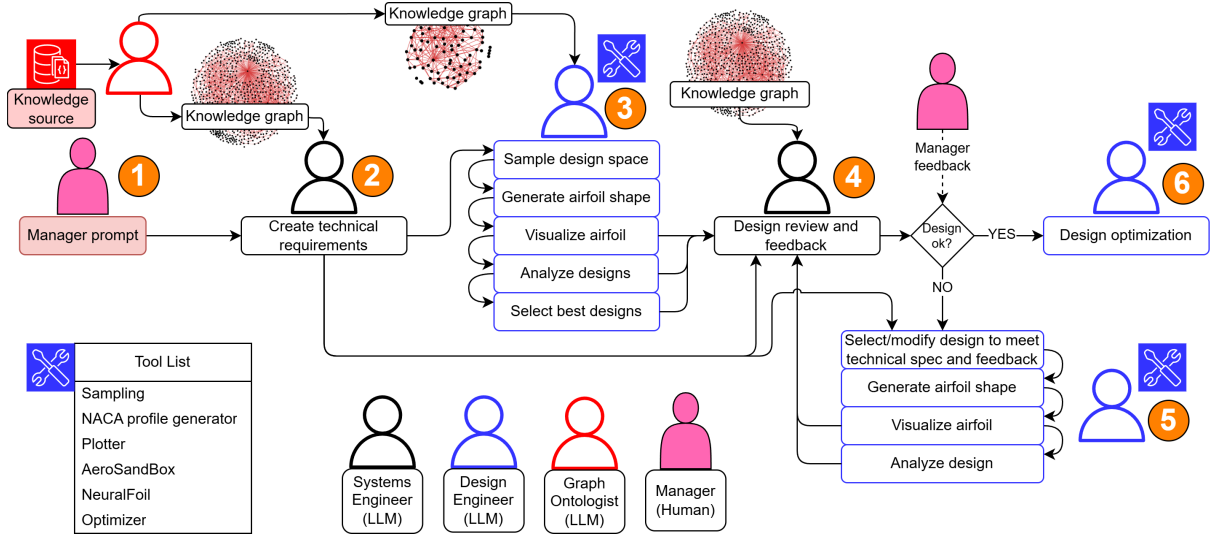
This work addresses the challenges and opportunities presented by the confluence of iterative design processes and the rise of LLMs by introducing a novel multi-agent framework for use in iterative design and review processes, particularly within the *conceptual* design phase. We focus on the pedagogical problem of designing 4-digit NACA airfoils for maximizing performance—a common and accessible exemplar problem in aerospace engineering—to demonstrate the capabilities of our multi-agent framework. The framework comprises three specialized AI agents, each assigned a distinct purpose and contributing to the overall design process: a *Graph Ontologist*, a *Design Engineer*, and a *Systems Engineer*. The Graph Ontologist acts as a knowledge curator, leveraging LLMs to extract and structure information from existing literature pertaining to airfoil design and development, helping to create two distinct knowledge graphs, specific for each engineering agent. These two graphs serve as source of background information, accessible to the designated agents within the framework. The Design Engineer agent utilizes the designer’s knowledge graph and design-specific tools to generate and refine airfoil designs,

guided by the knowledge contained within the graph. The Systems Engineer agent is tasked with translating the human manager’s request into formal technical requirements, and also reviews the designs generated by the Design Engineer agent, providing feedback for improvements.

The agents accomplish the design process through an iterative feedback loop. The Design Engineer agent proposes a design, the Systems Engineer agent evaluates it against the defined requirements and provides feedback, and the Design Engineer agent iterates on the design based on that feedback. This cycle repeats until the Manager (human) deems a design as acceptable in conjunction with the Systems Engineer qualifying the design as ‘valid’ against the requirement set. This approach aims to combine the knowledge-curation capabilities of LLMs with the rigor of established engineering practices, creating a powerful tool for iterative design and review. The complete workflow for this framework is shown in Figure 7.1.

## 7.2 Background and Related Works

Multi-Agent Systems (MAS) have long been proposed to support engineering design through distributed decision-making, concurrent workflows, and interdisciplinary coordination. Early work in the 1990s emphasized knowledge-based and protocol-driven agents for concurrent engineering. Foundational systems such as PACT demonstrated ontology-based coordination among agents for cooperative design refinement [230]. Similar MAS approaches were applied to cable harness design and distributed design environments, highlighting benefits of problem decomposition and message-based coordination but also exposing limitations due to hand-crafted rules and communication overhead [231, 232, 233]. Subsequent research explored alternative coordination mechanisms, including market-based agent marketplaces for concurrent, set-based design [234]. In conceptual design, agent-based configuration synthesis systems such as A-Design employed specialized agents to generate, evaluate, and refine electro-mechanical concepts using Pareto-based selection



**Figure 7.1:** Schematic for LLM-based multi-agent workflow, consisting of three LLM agents namely Graph Ontologist, Design Engineer, Systems Engineer, each assigned a designated set of tasks in this process. The Design and Systems Engineer agents are provided with their respective knowledge graphs tailored specific to their roles. The Design Engineer agent possesses a set of engineering tools to help him accomplish his tasks. The Systems Engineer agent utilizes a multi-modal language model for reviewing airfoil shapes generated by the Design Engineer and provides design improvement suggestions. The Design Engineer then acts on the feedback, updates the design, and sends it for another round of review process. This process also allows for a Manager (human user in this case) to provide feedback for design improvement. The design-review cycle terminates once the Manager determines the design to be a suitable candidate for further optimization. Details for each component and steps in this workflow are discussed later in Section 7.4.

[235, 236]. However, these approaches assumed well-defined objectives, which are often unavailable in early design stages.

With the maturation of CAD/CAE and PLM infrastructures, MAS research shifted toward distributed collaboration and interoperability. Agent frameworks supported manufacturability, cost, and lifecycle analysis by coordinating heterogeneous tools through facilitator agents and shared ontologies [237, 238, 239, 240]. Domain-specific systems addressed structural, process, and assembly design, as well as knowledge management and process coordination, but incurred high knowledge-engineering costs [241, 242, 150, 243, 244]. More recent work integrates MAS into CAD-adjacent pipelines, enabling multi-level 3D collaboration, sustainability-aware design, and automated analysis–optimization loops [245, 246, 247]. Knowledge-based and Belief-Desire-Intention (BDI)-style agents have further enabled explainable design evaluation and digital design reviews, reinforcing MAS

potential while highlighting ongoing challenges in scalable knowledge maintenance and geometric reasoning [248, 249, 250].

More recently, the rapid advancement of large language models (LLMs) has created a new research direction - development and use of LLM-powered MAS within engineering contexts. Before the rise of MAS frameworks, several studies have demonstrated the value of individual LLM assistants in engineering workflows [146, 251, 252, 253, 254]. Recent research ideology, however, prescribes MAS frameworks as being more suited for handling end-to-end design workflows, since different agent skill sets is desired at different stages of design. Early explorations of LLM-driven multi-agent systems such as DesignGPT [141] demonstrated a design workflow targeted at using generative AI image generation for generic product design. However, this framework does not address a specific engineering problem requiring domain knowledge and, hence, is prone to hallucinations. [255] and [142] propose MAS frameworks wherein user requirements, expressed as text or sketches, are translated into CAD models via coordinated LLM agents responsible for requirements engineering, geometric modeling, and vision-based verification.

[256] introduced a LLM assisted agent to construct and query a large ontological graph derived from 1000 scientific papers published in the area of biological material design. [152] further extended this to MAS, to autonomously generate and refine research hypothesis for new material discovery. [257] reviewed the path from LLM reasoning to autonomous agents, providing general design of agent tool-use, memory, and multi-agent coordination. [258] proposed the framework for AI collaboration in engineering design (AICED), where four LLM agents accomplish tasks associated with market research, documentation, design, and manufacturability using required design knowledge from professionals. In [144] a MAS framework was introduced to automatically perform tasks such as conceptual sketching, shape retrieval, and CFD meshing by integrating a set of agents within the automotive design workflow. [145] advanced LLM agentic workflows tailored to engineering design, by automating requirement gathering and evaluation criteria using design tools such as TRIZ [259] and QFD [260]. This framework utilized design loops to evaluate and refine



conceptual designs for the autonomous emergency response aircraft problem [261].

Mechanical Engineering Design Agents (MEDA) [262] introduced MAS for parametric CAD model creation, mapping design intents into CAD features via collaborating agents. In [263], a cost-aware MAS was proposed for design space exploration for search under expensive black-box objectives, to model the sequential-decision making process of a design team. This method uses a combination of Bayesian Optimization and Reinforcement Learning to distribute the task of design space exploration between multiple agents. [264] utilized vision-language models in an Generator-Critic framework for concept generation and evaluation where the Generator generates initial concept which is refined iteratively based on Critic feedback. The Critic provides both qualitative and quantitative feedback on design suitability. In [265], a multi-agent framework is used for conceptual design of solar-powered water filtration system, where dedicated agents perform functional decomposition, design generation, evaluation and grading in an iterative cycle. We note, however, that this framework relies on existing knowledge base of LLM agents and does not use human-in-loop ideology. [266] propose a multi-agent framework for designing autonomous water quality monitoring system, which involves agents representing mechanical designer, simulation, and electronics designer. The mechanical designer uses a set of code-based tools for ideation while the electronics designer uses knowledge acquired from technical data sheets for proposing suitable electronic components. At a higher level, [267] articulates a paradigm shift toward adopting agentic AI framework for automation of engineering processes.

Based on a review of existing literature, we observe that integrating MAS into engineering design workflows presents substantial challenges due to the inherently collaborative, iterative, and domain-knowledge intensive nature of engineering practice. Traditional design relies on coordinated expertise from multiple specialists—such as structural, materials, and manufacturing engineers—whose reasoning must either be encoded in large knowledge bases or learned from extensive datasets, both of which are resource-intensive [268, 269]. Effective MAS deployment further requires deep domain knowledge, including

design factors, manufacturing constraints, and regulatory requirements, which is difficult to represent in forms suitable for effective autonomous agent reasoning. Additionally, an ideal MAS system should be able to automatically adjust design parameters, rerun analysis, and flag potential design issues based on a specific set of knowledge. However, current systems often lack the sophisticated adaptive decision-making capabilities required for such nuanced problem-solving [270]. While multi-agent frameworks offer a promising avenue for encoding and managing such domain expertise, enabling them to interact with existing engineering tools and adapt to iterative design processes remains a research opportunity. Hence, the key objective of our work includes the following:

- Utilize LLM-agents equipped with existing domain-specific knowledge in an ‘Generator-Critic’ setup for fulfilling specific engineering roles.
- Integrate MAS into an iterative design framework where outcomes from one LLM agent assist another agent with its decision making and task completion.
- Associate dedicated engineering tools with agents to enable successful task completion.
- Allow human-in-loop interaction to ensure necessary check and balances in the process.

## 7.3 Problem Setting

The design of NACA airfoil profiles has historically played a crucial role in the advancement of aerodynamics in aviation [271]. The National Advisory Committee for Aeronautics (NACA) developed families of airfoils and categorized them based on their geometric and aerodynamic properties, with the 4- and 5-digit series being widely used. The significance of NACA airfoils predominantly lies in their predictable aerodynamic performances, making them instrumental in both academic research and practical applications [272]. Their predictable behavior simplifies the analysis of how airfoil shape affects

lift, drag, and overall aerodynamic efficiency, and serves as a pedagogical example for demonstrating our multi-agent design workflow.

The airfoil shape is defined by parameters such as maximum camber, maximum thickness, and maximum camber position. These parameters significantly influence the aerodynamic performance of the airfoil. Small variations in these parameters can lead to substantial changes in the pressure distribution around the airfoil, and thus, its overall aerodynamic performance [273]. For instance, increasing the maximum camber generally increases lift but also increases drag, while repositioning the maximum camber affects the pressure gradients and can influence stall characteristics. Understanding and precisely controlling these relationships is crucial for effective airfoil design. The four-digit NACA equation is foundational in the generation of these airfoil shapes [272], and is given by Equation 7.3.1:

$$y_t = 5t(0.2969\sqrt{x} - 0.1260x - 0.3516x^2 + 0.2843x^3 - 0.1015x^4), \quad (7.3.1)$$

where  $y_t$  is the half thickness at a given position  $x$  along the chord, and  $t$  represents the maximum thickness as a fraction of the chord. In this work, we consider the design space defined by airfoil parameters max-camber ( $c_{max}$ ), max-camber location ( $c_{loc}$ ), and max-thickness ( $t_{max}$ )

$$\mathcal{D} := [0.01, 0.095] \times [0.05, 0.9] \times [0.01, 0.40], \quad c_{max}, c_{loc}, t_{max} \in \mathcal{D}. \quad (7.3.2)$$

The central goal of this research is to design an efficient NACA airfoil profile, specifically targeting a maximum lift-to-drag ratio at Mach number 0.6 and Reynolds number  $5 \times 10^6$ . This specific operating condition represents a typical test condition for airfoil design validation [274]. Furthermore, for simplicity during the initial design phase, we assume a 0-degree angle of attack. This simplification allows us to focus on the fundamental

shape selection and optimization steps without the complexities introduced by the angle of attack. The design problem is chosen for its simplicity and the abundance of readily available technical documentation, which will be essential for constructing knowledge graphs in subsequent stages of the research. This problem definition can be adapted based on managerial requirements, but in the current scope, we concentrate solely on the aforementioned conditions. Furthermore, non-functional constraints will arise from the specific geometric parameters chosen for airfoil design. For example, excessively sharp leading edges, while potentially reducing drag, may be structurally impractical for manufacturing and operation. Such considerations are embedded as knowledge sources with our engineering agents, thereby enabling a holistic approach to design selection and evaluation process.

Furthermore, to address this optimization challenge, advanced tools such as AeroSandBox [275] and NeuralFoil [276] are utilized. AeroSandBox facilitates fluid simulations and aerodynamic analysis, while NeuralFoil offers machine learning capabilities to refine the design by anticipating aerodynamic characteristics of new airfoil shapes. These tools, when integrated, provide a robust framework for conducting design and analysis iteratively and efficiently. We discuss these tools in detail later in section 7.4. It is essential to acknowledge that while this investigation is focused on a specific aerodynamic design problem, the methodology employed here is adaptable to other design scenarios.

## 7.4 Methodology

### 7.4.1 Knowledge Graphs for Engineering Agents

To imbue our LLM agents with domain expertise and facilitate informed decision-making within the engineering design process, we leverage knowledge graphs (KGs) as a means of representing structured knowledge [154]. KGs offer a powerful and efficient way to encode a large body of existing knowledge, going beyond simple text-based information

retrieval [277, 278, 279]. The graph-based representation allows for the discovery of novel connections and ideas between concepts, even those that might appear distant based on human intuition but are ontologically linked. This ability to uncover hidden relationships is particularly valuable in complex engineering domains where tacit knowledge and interdependencies are prevalent. Our approach utilizes two distinct KGs, designed to represent differing levels of knowledge abstraction and, consequently, to simulate varying degrees of expertise within our LLM agents, mirroring the spectrum of expertise found in real-world engineering practice.

These KGs were generated using a dedicated Graph Ontologist agent [256], which operates according to a set of pre-defined instructions tailored to create agent-specific knowledge graphs. This controlled generation process ensures consistency and relevance of the knowledge encoded within each graph. The content for these graphs is derived from a comprehensive corpus of published reports and papers focusing on NACA airfoil design and development. This collection spans a considerable period, from the early years of NACA research in 1940s to more recent publications in 2010s, effectively capturing the evolution of knowledge and best practices accumulated over decades. This long-term perspective is crucial for mimicking the experience and understanding that an expert engineer develops over years of training and practical application. It avoids the pitfalls of relying solely on the latest information, ensuring that fundamental principles and historical context are also incorporated into the agents' knowledge base.

The use of KGs within our LLM agents provides a crucial foundation for ensuring that design choices are supported by consistent practices and prior findings. Without this structured knowledge, agents would be prone to generating inconsistent or suboptimal designs and responses. Furthermore, the KGs serve as a mechanism for transparency and explainability, allowing us to trace the origins of design recommendations and understand the reasoning behind agent decisions [280]. This capability is essential for building trust and facilitating human oversight within the design process. In the subsequent section, we will detail the processes involved in knowledge graph creation and how the Graph

Ontologist agent constructs these resources.

### 7.4.2 Graph Ontologist Agent

To facilitate the creation of specialized knowledge graphs for our Design and Systems Engineering agent, we employed a dedicated Graph Ontologist agent. This LLM agent was specifically designed to extract and structure knowledge from unstructured text, forming the foundation of our knowledge graph. The Graph Ontologist agent utilizes LLM capabilities to interpret context and infer semantic relationships. The corpus utilized for knowledge graph generation comprises of 50 scientific articles and reports spanning decades of NACA airfoil design and development. Initially, the raw data from the scientific papers was distilled through a sequence of preparatory steps. This included converting the PDF into a markup language format, followed by segmenting the text into manageable chunks. A chunk size of 3000 was chosen heuristically to represent semantically coherent units of information, hence allowing the Graph Ontologist agent to focus on localized knowledge extraction. Each text chunk was then processed by the Graph Ontologist agent to extract local knowledge graph triples, representing relationships between key concepts. These triples consisted of two nodes, each representing a significant concept from the extracted ontology, and an edge feature that defined the relationship between the two concepts. The triple set is defined as<sup>1</sup>: ‘node1’: “A concept from extracted ontology”, ‘node2’: "A related concept from extracted ontology”, ‘edge’: “relationship between the two concepts, node1 and node2 in one or two sentences”. This explicit definition guided the agent’s extraction process, ensuring a consistent and interpretable knowledge representation. The procedure for KG generation is adopted from [153].

We employ two KGs in this study that differ significantly in scope and detail. The first KG, designed for the Systems Engineer agent, encompasses a broad range of aspects related to airfoil and aircraft development. This includes design considerations, fabrication

---

<sup>1</sup>Code for knowledge graph generation was obtained from [https://github.com/raahulnyk/knowledge\\_graph](https://github.com/raahulnyk/knowledge_graph)

processes, validation procedures, and experimental evidence. This KG represents the perspective of a seasoned systems engineer with extensive experience overseeing the development of aerodynamic systems across various product design contexts. The relationships within this graph emphasize the interdependencies between different stages of the design lifecycle and the impact of design decisions on overall system performance. The second KG, utilized by the Design Engineer agent, adopts a more focused perspective on specific design aspects influencing airfoil shapes and their subsequent aerodynamic performance. This graph primarily focuses on the relationships between design variables, their impact on key performance metrics, and typical values observed in practical applications. The focus is on creating a more granular knowledge base for the Design Engineer agent. To obtain these distinct KGs, two sets of prompts were provided to the Graph Ontologist agent as shown in Figure 7.2. The Mistral-NeMo model was utilized iteratively to extract these triples from the text chunks, leveraging its ability to understand complex sentence structures and identify subtle semantic relationships.

The individual local graphs generated from each text chunk were subsequently consolidated into a global knowledge graph. This process involved merging nodes representing similar concepts and eliminating ‘hanging’ nodes—those that were not connected to the rest of the graph, indicating potentially erroneous extraction. The merging of similar nodes was performed using a similarity metric based on node embeddings, ensuring that semantically related concepts were integrated into a unified representation. The edge weights were increased during the concatenation process, reflecting the cumulative evidence supporting the relationships between concepts. The resulting knowledge graphs for the two agents are visually represented in Figure 7.3, illustrating the distribution of nodes and edges within each graph. The structural differences between the graphs, as reflected in their visual representation, highlight the impact of the different prompts used to guide the Graph Ontologist agent, demonstrating the adaptability and utility of this approach for creating specialized knowledge resources.

| Graph Ontologist Prompt for Systems Engineer's KG                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | Graph Ontologist Prompt for Design Engineer's KG                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>"You are a network graph maker who extracts terms and their relations from a given context. You are provided with a context chunk related to airfoils design and analysis. Your task is to extract the ontology of terms mentioned in the given context. These terms should represent the key concepts related to airfoil design, development and analyses as per the context. Pay special attention to relationships that describe airfoil parameters such as camber, camber location or thickness, values of these parameters and how they affect performance such as lift, drag and manufacturability."</p> | <p>"You are a network graph maker who extracts terms and their relations from a given context. You are provided with a context chunk related to airfoils design and analysis. Your task is to extract the ontology of terms mentioned in the given context. These terms should represent the key concepts related to airfoil design, analyses and optimization only. Pay special attention to relationships that describe airfoil parameters such as camber, camber location or thickness, values of these parameters and how they affect performance such as lift, drag and manufacturability. Ignore names of people, authors, their organizations when performing this evaluation. You should also ignore abstract items that do not necessarily relate to airfoil shape designs. You should simplify abbreviations when you think they need to be defined and are specific to the context you are analyzing."</p> |

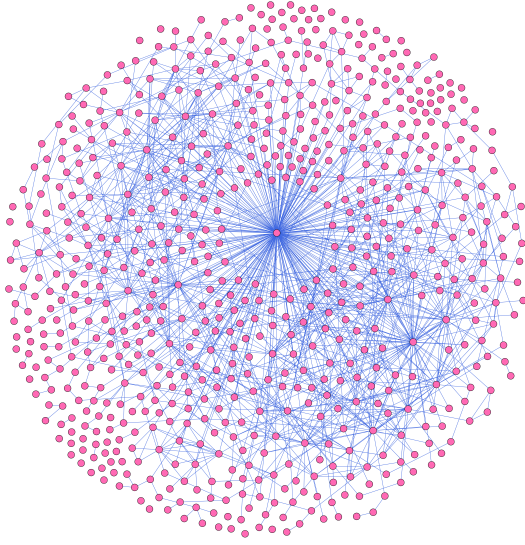
**Figure 7.2:** Prompts provided to Graph Ontologist for generating two different KGs for our engineering agents. For the Systems Engineer KG, a more generic set of goals leads to the creation of a larger domain of knowledge as compared to the Design Engineer's KG, which is more focused on aspects related to airfoil design.

### 7.4.3 Systems Engineer Agent

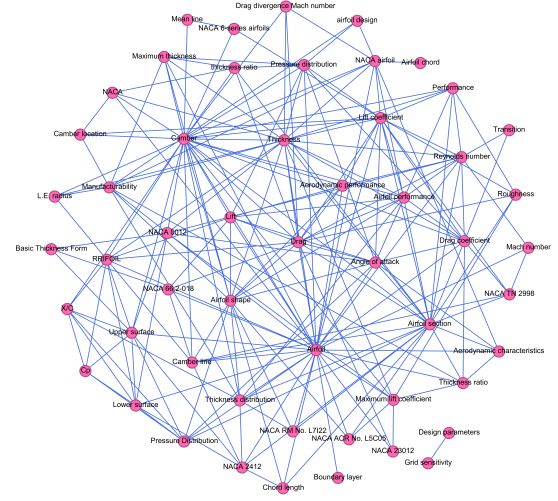
The Systems Engineer agent embodies the responsibilities of a senior systems engineer, leveraging a comprehensive knowledge base and advanced reasoning capabilities to generate technical requirements, review designs, and offer recommendations for design improvements based on knowledge. The design of this agent is informed by established practices in systems engineering, where a systems engineer acts as an integrator, ensuring alignment between technical specifications, design choices, and overall project goals. The Systems Engineer Agent operates within a layered architecture, integrating several key components: a knowledge source, a prompt engineering module, and an LLM execution engine.

The agent's effectiveness is directly tied to the quality and accessibility of its knowledge





(a) Knowledge graph used as knowledge base for Systems Engineer agent.



(b) Knowledge graph used as knowledge base for Design Engineer agent.

**Figure 7.3:** Representation of Knowledge Graphs generated by the Graph Ontologist for the two engineering agents. Systems Engineer’s KG contains  $\approx 700$  nodes while the Design Engineer’s KG contains  $\approx 50$  nodes. Note that these graphs only contain nodes with a degree of 10 or more, the remaining nodes were filtered out for compactness of the knowledge base.

base. This knowledge source represents the collective experience and expertise accumulated over years of practice. We represent this knowledge in the form of a knowledge graph, implemented as a CSV file containing nodes and relationships. This representation allows for structured storage and retrieval of information, facilitating informed decision-making by the agent. To manage the scale of the knowledge graph, the CSV content is split into manageable chunks, each associated with metadata such as chunk ID, source reference, and row range [281]. This chunking strategy optimizes the embedding and retrieval process. Text chunks are then processed using an embedding model (OpenAI’s text-embedding-3-small) to generate vector representations [282]. These vectors are stored in a vector database for efficient similarity search. This vector database approach enables rapid identification of relevant knowledge based on semantic similarity to the user’s query [283]. The use of vector embeddings allows the agent to understand the underlying meaning of the query and retrieve more contextually relevant information.

At runtime, the Systems Engineer Agent converts user prompts into “search queries”

through the prompt engineering module. This conversion often leverages the LLM itself to rephrase or expand upon the original prompt, ensuring that it effectively captures the user’s intent. The refined query is then used to search the vector database, retrieving the top-ranked and most relevant chunks. Thereafter, the retrieved chunks are appended to the user’s original prompt, providing the agent with the necessary context to generate a comprehensive and informed response [282]. This process mimics the way a human systems engineer would consult internal documentation and prior experience to address a new design challenge. In Appendix A3.1, we empirically demonstrate examples of knowledge retrieval from the graph for verifying the RAG mechanism during the ‘Requirement Generation’ step.

The Systems Engineer Agent performs several critical tasks within this framework. These include:

- **Technical Requirements Generation:** Based on user prompts and leveraging its knowledge source, the agent generates technical requirements. The contextual role is provided to the LLM agent to accomplish this task, ensuring consistency and adherence to established engineering principles. This capability streamlines the requirements elicitation process and minimizes the potential for ambiguity or omissions. One key advantage with using knowledge source for this task is the ability to formulate both functional and non-functional requirements simultaneously.
- **Airfoil Design Review:** The agent evaluates new airfoil designs, comparing them against established technical requirements and utilizing its accumulated design knowledge. The agent receives the airfoil profile generated by the Design Engineer along with the corresponding aerodynamic performance results. Critically, the agent incorporates multi-modal vision models to assess airfoil profile plots and performance data visually. This capability allows for a more nuanced evaluation than relying solely on numerical data, accounting for factors such as visual symmetry, smoothness, and overall aesthetic appeal. Feedback is provided to the Design Engineer, highlighting

potential areas for improvement and suggesting alternative design approaches.

The agent’s feedback on airfoil designs is driven by a dual consideration: alignment with technical requirements and leveraging its knowledge base for prior experience with design specifics. This knowledge base represents the accumulated understanding of design trade-offs, potential failure modes, and best practices. The agent uses this knowledge to anticipate potential issues, suggest proactive design improvements, and ensure that the final design meets the required performance characteristics. The agent uses Google’s ‘gemini-2.5-flash’ LLM model for its task accomplishment.

#### **7.4.4 Design Engineer Agent**

The Design Engineer agent represents a critical mid-level role within our multi-agent system designed to automate and enhance the airfoil design process. This agent is responsible for the iterative cycle of creating, analyzing, visualizing, and refining airfoil designs. The Design Engineer agent collaborates closely with the Systems Engineer agent to ensure design feasibility and adherence to overall system requirements to achieve performance targets. The core functionality of the Design Engineer agent centers around managing the design space, generating candidate airfoils, analyzing their aerodynamic performance, and iteratively refining designs based on feedback. This process is not a linear sequence but a continuous loop, with insights gained from analysis triggering adjustments to the design and further rounds of evaluation. Below we describe the core tasks and tools associated with this agent.

##### **Design Space Sampling**

The agent commences its task by sampling the design space, utilizing a suite of strategies to generate diverse candidate airfoils. The *Sampling tool*, a custom-built Python function, provides options for Latin Hypercube, Sobol, Halton, and random sampling. Each strategy possesses unique characteristics suitable for different design exploration

goals. The tool generates a set of parameter values for maximum camber, maximum camber location, and maximum thickness that define the 4-digit NACA airfoil geometry.

### **NACA Profile Generation**

Once the design parameters are sampled, the *NACA profile generator tool* utilizes the standard 4-digit NACA airfoil equation to generate the corresponding airfoil coordinates. This equation, based on a mathematical representation of airfoil geometry, enables the creation of a wide variety of airfoil shapes by adjusting three key parameters: the location of the maximum camber, the maximum camber value, and the maximum thickness value. It is important to note that these initial designs are not intended to be final products but rather starting points for further refinement.

### **Airfoil Visualization**

The airfoil shapes generated are then visualized using the *Plotting tool*, a Python function using airfoil coordinates to plot its shape. These visualizations are critical for review by the Systems Engineer agent, allowing for a qualitative assessment of each design against technical requirements, especially non-functional requirements.

### **Aerodynamic Analysis with AeroSandbox and NeuralFoil**

Following visualization, the aerodynamic performance of the generated designs is evaluated using the *AeroSandbox tool*, an open-source, Python-based framework [275]. AeroSandbox uniquely integrates physics-based aerodynamic models, structural modeling, and multidisciplinary optimization within a unified computational environment. Unlike conventional CFD tools, AeroSandbox leverages low-fidelity aerodynamic models, such as vortex lattice methods and thin airfoil theory, coupled with automatic differentiation capabilities through tools like CasADi. This approach facilitates rapid design space exploration and gradient-based optimization of airfoils, wings, and complete aircraft configurations.

Complementing AeroSandbox is *NeuralFoil*, a machine-learning-based airfoil analysis tool that utilizes a neural network trained on extensive XFOIL simulation data [276]. NeuralFoil’s predictive capabilities for aerodynamic coefficients ( $C_l, C_d, C_m$ ) span a broad spectrum of Reynolds numbers and angles of attack, enabling near-instantaneous predictions with acceptable accuracy. By substituting numerical flow solutions with learned surrogates, NeuralFoil offers substantial computational savings, making it particularly well-suited for real-time applications, optimization loops, and scenarios where CFD simulations are impractical. This tool is primarily used for design analysis and optimization.

## Design Optimization

Design optimization represents the final step in the design process where a candidate design chosen during the iterative design phase is optimized against the design objectives. The *Optimization tool* utilizes Kulfan coordinates to represent and manipulate the airfoil geometry during the optimization stage [284]. Kulfan coordinates offer a distinct advantage over traditional Cartesian coordinates by providing a smoother and more stable representation of the airfoil shape, particularly when dealing with significant geometric changes. This property facilitates the optimization process by preventing abrupt shifts and ensuring a more controlled evolution of the design. The optimizer employs a multi-objective optimization routine, balancing aerodynamic performance metrics such as lift and drag coefficient, while adhering to other general constraints required for convergence.

## Design Selection and Revision

The designs generated by the Design Engineer agent are reviewed by the Systems Engineer agent in an iterative process. The Design Engineer, upon receiving critical feedback about the designs then initiates a design selection and revision process, using his background knowledge and feedback received. This agent then modifies the design appropriately, and provides the new design sample for review again.

The Design Engineer agent plays a critical role in iteratively refining designs based on Systems Engineer’s feedback, technical requirements, and its own knowledge base. For example, if the Systems Engineer suggests a reduction in thickness value to aid with manufacturing, the Design Engineer might choose a lower maximum thickness value for generating the new airfoil shape guided by his knowledge. Such domain knowledge driven interaction between agents generates context-aware suggestions, considering not only the immediate performance goals but also the broader system requirements and manufacturing limitations. This integration allows the Design Engineer to operate with a higher level of intelligence and adaptability by continuously refining the design towards optimal performance and feasibility. The Design Engineer agent uses OpenAI’s ‘gpt-4o’ LLM model for its task accomplishment.

#### **7.4.5 Manager (Human User)**

Our framework incorporates a Manager role, representing the human user responsible for guiding the design exploration process. This individual does not directly manipulate the LLM-based design agents, but instead acts as a gatekeeper and evaluator, ensuring the generated designs align with overarching project goals, constraints, and evolving requirements. The Manager’s role is crucial for injecting domain expertise and addressing nuances that automated systems may overlook. The Manager’s involvement begins at the problem definition stage. The initial design problem is formulated and communicated to the LLM-based design agents, setting the scope for subsequent exploration. During the final design selection phase, the Manager critically assesses a limited set of candidate designs, typically the top-ranked solutions chosen by the Design Engineer agent. This evaluation considers factors not easily quantifiable by the automated system, such as aesthetic preferences, manufacturability, and potential for future scalability. The Manager’s authority extends to determining when the design cycle concludes. This termination condition is triggered by the Manager’s explicit acceptance of a design, indicating a

satisfactory outcome has been achieved. Furthermore, the Manager’s ability to terminate the process allows for agile adaptation to changing project needs or the identification of fundamentally flawed design approaches. This provides a mechanism for course correction and ensures that the design process remains responsive to evolving requirements. Thus, this human-in-the-loop framework allows us to harness the power of LLM-based agents while retaining the crucial element of human expertise and judgment, facilitating a more robust and adaptable engineering design process.

#### **7.4.6 Workflow**

In this section, we describe the sequence of operations in our multi-agent workflow shown in Figure 7.1. The design workflow within this multi-agent system is structured around a main design sequence, orchestrated by the Manager, Systems Engineer, and Design Engineer agents, alongside an auxiliary sequence managed by a Graph Ontologist agent. The auxiliary knowledge graph (KG) generation sequence is executed prior to the main design sequence, establishing a foundational knowledge base for subsequent design iterations. This sequential approach aims to leverage the strengths of each agent with the Manager (human user) providing high-level direction, the Systems Engineer translating requirements, the Design Engineer generating and refining designs, and the Graph Ontologist providing structured knowledge. The main design sequence consists of six steps:

1. Kickoff: The workflow initiates with a kickoff phase where the Manager provides an initial prompt to the Systems Engineer agent. This prompt outlines the problem goals and objectives, serving as the genesis of the design process. The initial prompt provided by the Manager is:

#### Manager prompt for Kickoff

“Develop a 4-series NACA airfoil that has the maximum lift to drag ratio at Mach=0.6. You can assume  $Re = 5 \times 10^6$  approximately and an AoA=0 deg for this phase of development.”

The selection of Mach=0.6,  $Re=5 \times 10^6$ , and AoA=0 represents typical test conditions for a range of applications, facilitating a relevant and focused design exploration .

2. Requirement elicitation phase: Following the kickoff, the Systems Engineer agent undertakes requirement elicitation task. This crucial step transforms the initial prompt into a more detailed and technically grounded set of engineering requirements. The KG plays a pivotal role here, leveraging existing knowledge to elaborate on vague objectives and translate them into quantifiable design targets. For example, a general requirement like “design needs to minimize drag” might be transformed into “design must have a  $C_d < 0.01$ ” through the KG’s structured knowledge. Functional requirements, such as the aforementioned drag coefficient limit, can be directly assessed and evaluated. Non-functional requirements, like “the airfoil appearance must be aesthetically pleasing to the target customer,” are essential for comprehensive design consideration, though inherently more challenging to quantify and may involve qualitative assessments using our LLM agents. To illustrate this transformation, consider the initial prompt concerning maximum lift-to-drag ratio. The KG, enriched with aerodynamic principles and design best practices, can elicit specific functional requirements such as a target  $C_l$  range or a maximum allowable  $C_m$  value. Non-functional requirements might include constraints on manufacturing feasibility or aesthetic preferences.
3. Design phase: The subsequent design phase is primarily managed by the Design Engineer agent. This involves the following sub-phases:
  - (a) Design space sampling: A stage where the agent generates a set of design



parameters: maximum camber, maximum camber location, and maximum thickness, based on a predefined strategy. We selected a sample size of 100 designs, justified by the need to explore a reasonable portion of the design space without overwhelming the subsequent analysis and review processes.

- (b) Airfoil shape generation: The chosen design parameters are then fed into the NACA profile generator tool, producing airfoil profiles. The tool, by default, generates profiles with 200 points on both the top and bottom surfaces.
- (c) Airfoil visualization: Following generation, the plotting tool is employed to create visual representations of these profiles, facilitating intuitive understanding and communication throughout the design process.
- (d) Airfoil analysis: The generated airfoil shapes are then subjected to aerodynamic analysis using NeuralFoil and AeroSandbox as discussed earlier. The primary Quantities of Interest (QoIs) assessed include lift coefficient ( $C_l$ ), drag coefficient ( $C_d$ ), and pitching moment coefficient ( $C_m$ ).
- (e) Filtering: Following analysis, a filtering step is implemented. Airfoils with  $C_l < 0.5$  are filtered out; this criterion reflects a practical engineering constraint related to the minimum lift requirement for stable flight and avoids designs that would be non-viable for intended applications. The top 10 ranking designs are then progressed for design review, effectively reducing the burden on the Systems Engineer and aligning with time constraints commonly encountered in real-world engineering teams.

4. Design review and feedback: The design review and feedback phase leverages the capabilities of ‘gemini-2.5-flash’ multi-modal model. The multi-modal input capability is critical; it allows the Systems Engineer agent to analyze both the visual representation of the airfoil profiles and their corresponding performance data. For example, a mutli-modal LLM can be used to assess the smoothness of the airfoil’s leading edge or to correlate a sudden increase in  $C_d$  with a specific visual anomaly

in the profile. The Manager can also provide his feedback at this stage, aligning the design with broader project goals. This feedback is integrated into the design process, guiding the Design Engineer’s subsequent modifications.

5. Design selection and revision: Following the review, the Design Engineer selects the most promising design candidate, incorporating design change feedback from the Systems Engineer. This design then undergoes further review and modification in an iterative cycle until deemed valid by the Manager. The final design candidate is then passed to an optimization tool embedded within the NeuralFoil framework.
6. Optimization: The optimizer utilizes objectives and constraints derived from the technical requirements and pre-configured options to converge on an optimal solution. The optimization algorithm and specific objectives are crucial factors influencing the quality and efficiency of the final design.

## 7.5 Experiments

This section details the key findings from our workflow, demonstrating the integrated functionality of LLM agents and human-in-the-loop interaction in an airfoil design process. We outline the sequence of events, from the initial generation of technical requirements to the final optimization of the airfoil shape.

### Creation of Technical Requirements

The design process commenced with Stage 2, the creation of technical requirements. Guided by a manager prompt, the Systems Engineer agent synthesized a comprehensive technical requirements document, as shown below. This document serves as a pivotal guiding principle for the entire team, outlining the functional and non-functional criteria the airfoil design needs to satisfy. The Systems Engineer agent’s ability to draw upon its knowledge base, presumably informed by established engineering principles and design standards, ensures that the technical requirements were both relevant and complete.

This initial step establishes a crucial foundation, minimizing ambiguity and setting clear objectives for the subsequent design phases.

### Functional Requirements

- "- Objective Function: The primary objective is to maximize the lift-to-drag ratio ( $Cl/Cd$ ) of the airfoil section.
- Airfoil Family: The airfoil design must strictly adhere to the NACA 4-digit series geometry.
- Geometric parameters:
  - Maximum camber ( $m$ ): The first digit of the NACA 4-digit series, representing the maximum camber as a percentage of the chord. The design space for 'm' shall be constrained to  $0\% \leq m \leq 9\%$ . A non-zero camber is expected to generate lift at  $AoA=0$ .
  - Location of maximum camber ( $p$ ): The second digit of the NACA 4-digit series, representing the position of maximum camber from the leading edge in tenths of the chord. The design space for 'p' shall be constrained to  $0 \leq p \leq 9$  (0% to 90% chord).
  - Maximum thickness ( $t$ ): The last two digits of the NACA 4-digit series, representing the maximum thickness as a percentage of the chord. The design space for  $t$  shall be constrained to  $6\% \leq t \leq 20\%$  to manage drag at  $M=0.6$  and avoid excessive drag coefficients.
- Operating conditions:
  - Mach Number ( $M$ ): The design must be optimized for a free-stream Mach number of 0.6.
  - Reynolds Number ( $Re$ ): The design must be optimized for a Reynolds number of approximately  $5 \times 10^6$  (based on chord length)
  - Angle of Attack ( $AoA$ ): The design must be optimized for an angle of attack of 0 degrees.
- Performance metrics: The design must provide quantified values for the lift coefficient ( $Cl$ ), drag coefficient ( $Cd$ ), and the resulting lift-to-drag ratio ( $Cl/Cd$ ) at the specified operating conditions."

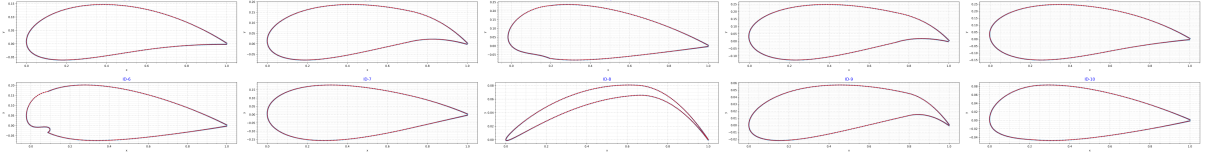
### Non-Functional Requirements

- "- Manufacturability: The resulting airfoil geometry should be practical and feasible for manufacturing using conventional methods, avoiding features that would significantly increase production complexity or cost.
- Aerodynamic performance
  - Critical Mach Number: The designed airfoil should aim to have a critical Mach number greater than 0.6 to minimize the onset of significant compressibility effects (e.g., wave drag) at the operating Mach number. The designed airfoil should aim to have a critical Mach number greater than 0.6 to minimize the onset of significant compressibility effects (e.g., wave drag) at the operating Mach number.
  - Boundary Layer Characteristics: The design must consider the boundary layer transition and separation characteristics at  $Re = 5 \times 10^6$  to minimize profile drag and ensure stable flow.
  - Pressure Distribution: The airfoil geometry should result in a favorable pressure distribution along the chord that contributes to high lift and low drag at  $AoA=0$  and  $M=0.6$ .
- Analysis Compatibility: The airfoil geometry must be compatible with standard computational fluid dynamics (CFD) or panel method tools (e.g., XFOIL) for accurate aerodynamic performance evaluation.
- Robustness: The design should ideally exhibit a reasonable performance envelope, meaning that minor deviations from the specified operating conditions ( $M=0.6$ ,  $Re=5e6$ ,  $AoA=0$ ) do not lead to drastic performance degradation."

## Initial Design Generation and Visualization

Stage 3 then initiated the design generation phase. The Design agent employed Latin Hypercube sampling to generate an initial set of 100 diverse airfoil designs. The Design agent then proceeded to generate and visualize the airfoil shapes themselves. This visualization step, as depicted in Figure 7.4, allows for an intuitive understanding of the design variations and facilitates visual assessment by Systems Engineer agent and Manager

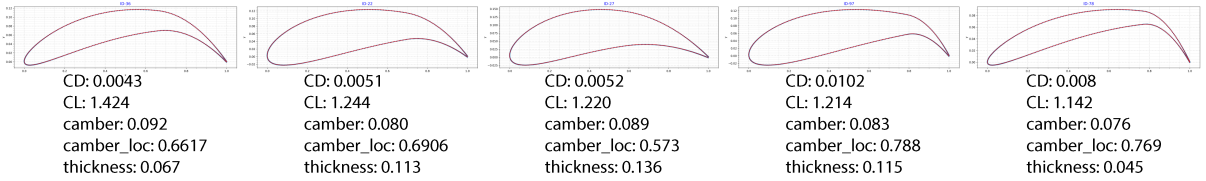
during the design review phase.



**Figure 7.4:** Initial airfoil samples generated by the Design Engineer agent using the NACA generator and plotter tool. The image shows 10 out of the 100 samples generated in this step following parameter selection in Stage 2.

### Design analysis and filtering

Following the generation of initial designs, a design analysis and filtering process was implemented to reduce the number of designs for the initial review. Each design was evaluated for its aerodynamic performance using NeuralFoil. Following this, the Design agent excluded designs exhibiting a coefficient of lift ( $C_l$ ) below 0.5, a guideline established to ensure basic aerodynamic performance. This filtering step is accomplished by a python function that allows designs to be sorted and filtered. Figure 7.5 shows the airfoil samples selected for design review. We note that airfoils with low thickness values exhibit higher lifts and consequently appear at the top of this filtered list. However, due to difficulties in meeting non-functional requirements such as manufacturability, some of these designs are subsequently considered invalid during the review process (by the Systems Engineer).



**Figure 7.5:** Initial filtered design samples selected for design review by the Systems Engineer agent.

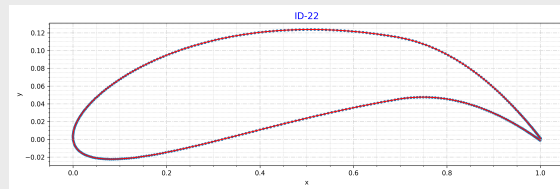
### Initial Design Review

In stage 4, the Systems Engineer agent leverages its expertise derived from a larger knowledge base along with a vision LLM to evaluate the proposed designs against the previously defined technical requirements. This evaluation involves a nuanced consideration of aerodynamic performance and adherence to the technical specifications, especially non-functional requirements that are qualitative. The Systems Engineer provides clear feedback on each design, categorizing them as either “invalid” (unable to meet technical requirements)

or “valid”. This structured feedback mechanism is crucial for guiding the Design Engineer agent’s subsequent revisions and ensuring that the design process converges on a good design candidate. Below we show feedback generated by the Systems Engineer agent for initial design samples from Stage 4. Additionally, in Appendix A3.2, we provide an example of the background reasoning and task breakdown used by the Systems Engineer agent during the design review process of a sample airfoil profile and aerodynamic data.

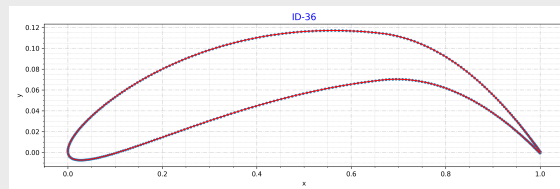
### Feedback on Design ID-22

"The airfoil profile (ID-22) visually presents a smooth, cambered shape with conventional leading and trailing edges, indicative of a well-formed NACA 4-digit series airfoil. The aerodynamic data shows exceptional performance with a lift coefficient ( $C_l$ ) of 1.244 and a very low drag coefficient ( $C_d$ ) of 0.00508, resulting in an outstanding lift-to-drag ratio ( $C_l/C_d$ ) of 244.88 at the specified operating conditions ( $M = 0.6$ ,  $Re = 5 \times 10^6$ ,  $AoA = 0$ ). The geometric parameters camber (0.08015), camber location (0.69065), thickness (0.11287) are all within the defined design space and contribute effectively to this high performance. To potentially improve the design further, consider exploring a slightly more forward camber location (e.g., 40-50% chord) as moving camber towards the leading edge can increase lift, which might further enhance the  $C_l/C_d$  ratio. Additionally, a detailed pressure distribution analysis and critical Mach number assessment would provide deeper insights for optimization.", "Valid":true



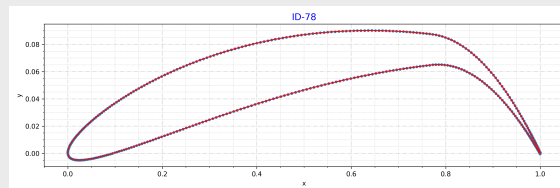
### Feedback on Design ID-36

"The airfoil profile, as depicted in the image, shows a well-defined shape with a noticeable camber, consistent with a lifting airfoil. The current design exhibits excellent aerodynamic performance with a very high lift coefficient ( $C_l = 1.42$ ) and an exceptionally low drag coefficient ( $C_d = 0.0043$ ), resulting in an outstanding lift-to-drag ratio of 330.79 at the specified operating conditions ( $M = 0.6, Re = 5 \times 10^6, AoA = 0$ ). The geometric parameters are: camber: 9.2%, camber location: 66.2% chord, and thickness: 6.7% chord. While the thickness and camber location are within the specified design space, the maximum camber of 9.2% slightly exceeds the functional requirement's upper limit of 9%. To strictly adhere to the NACA 4-digit series geometry constraint, the maximum camber should be adjusted to be at most 9%. Further optimization could explore moving the camber location slightly forward (e.g., 40-50% chord) to potentially enhance lift characteristics or refine pressure distribution, and a critical Mach number analysis is recommended to confirm compliance with the non-functional requirement.", "Valid":false



### Feedback on Design ID-78

"The airfoil profile ID-78, visually appearing very thin and cambered, demonstrates exceptional aerodynamic performance with a high lift coefficient ( $C_l = 1.14$ ) and a very low drag coefficient ( $C_d = 0.0085$ ), resulting in an outstanding  $C_l/C_d$  ratio of 134.45. This performance is largely attributed to the significant camber (7.62%) at an aft location (76.97% chord). However, the design is critically flawed as its maximum thickness (4.59%) falls below the minimum functional requirement of 6% (design space for 't' is  $6\% \leq t \leq 20\%$ ). This extreme thinness, while potentially contributing to a high critical Mach number, raises significant concerns regarding manufacturability and structural integrity. To improve the design, the maximum thickness should be increased to at least 6%, ideally targeting 6-8%, to comply with the requirements while maintaining good aerodynamic efficiency.", "Valid":false



## Iterative Design Revision and Review

Based on the Systems Engineer's feedback and its design-related knowledge base, the Design Engineer agent selects one design for further refinement. The Design Engineer then modifies the design parameters according to the Systems Engineer's suggestions, visualizes the resulting airfoil, and evaluates its performance using the AeroSandBox tool. This

chosen design is then presented to the Systems Engineer agent along with Manager for an iterative review-revision process. The response from the agent below shows the extracted knowledge from the design KG, reasoning used behind the choice of best candidate design and the revisions made to the design.

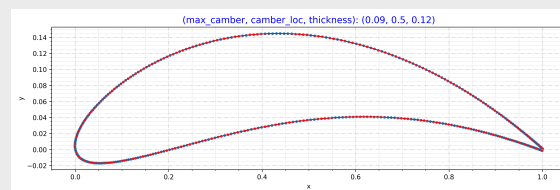
### Design revision 1: Designer's reasoning

**\*\*Design Parameters:\*\*** - Max Camber: 9% (0.09) - Camber Location: 50% (0.5) - Thickness: 12% (0.12)

**\*\*Reasoning:\*\*** - The maximum camber is set to 9% to adhere to the functional requirement's upper limit while maximizing lift. - The camber location is moved to 50% chord to potentially enhance lift characteristics, as suggested by feedback and supported by the knowledge base. - The thickness is set to 12% to ensure manufacturability and maintain structural integrity, while also being within the specified design space.

**\*\*Knowledge Source Reference:\*\*** - *Moving the camber location towards the leading edge increases lift.* - *Higher camber values can increase the lift coefficient but may also increase drag due to flow separation.* - *Thickness affects drag and manufacturability.*

The revised airfoil design has been generated using the 'generate airfoil' tool with the specified parameters.



This is followed by a design review jointly by the Systems Engineer and the Manager. The feedback provided by the Systems Engineer is shown below, where the design specification provided by the Designer is compared against the Functional requirement of meeting the NACA profile standards. Additionally, the Systems Engineer also identifies areas of improvement such as adjustments to camber location to help improve the pitching moment.

### Design revision 1: Systems Engineer feedback

"The airfoil profile, identified as NACA 9512 from the geometric parameters (camber: 0.09, camber location: 0.5, thickness: 0.12), exhibits a smooth, cambered shape typical of the NACA 4-digit series. The aerodynamic performance is exceptional, with a high lift coefficient ( $C_l = 1.135$ ) at  $AoA=0$  and a very low drag coefficient ( $C_d = 0.0055$ ), yielding an outstanding lift-to-drag ratio of 206.24. All geometric parameters adhere to the specified design space constraints. For further improvement, it is critical to verify the critical Mach number is indeed greater than 0.6 to avoid compressibility effects at the operating Mach number. Additionally, optimizing the camber location could help manage the pitching moment ( $CM = -0.272$ ) and a detailed boundary layer analysis could further refine drag reduction.", "Valid": true

At this point, the Manager intervenes in the review process and provides his/her feedback about the proposed design. This human-in-loop arrangement allows a human to supervise the design process, while the autonomous agents perform a major chunk of work.

### Design revision 1: Manager feedback

"I want the design to have standard camber of between 1-6% while maximizing L/D ratio and optimizing CM. Can you update your response accordingly and mark this design as invalid"

Based on this human feedback, the Systems Engineer updates its feedback for the Design Engineer with the updated requirement, while flagging the design as invalid to continue design iteration.

### Design revision 1: Systems Engineer updated feedback

"Thought: *The user has provided new feedback for the design. I need to incorporate this feedback into my analysis and update the 'Valid' status to 'False' for the current design ('revision1'). The key changes are:*

***\*\*Camber constraint\*\***: The design should have a standard camber between 1-6%. The current design has a camber of 9%. This violates the new constraint.*

The airfoil profile, identified as NACA 9512 from the geometric parameters (camber: 0.09, camber location: 0.5, thickness: 0.12), exhibits a smooth, cambered shape typical of the NACA 4-digit series. However, this design is **\*\*invalid\*\*** as the maximum camber of 9% violates the updated user requirement for camber to be between 1% and 6%. While the aerodynamic performance ( $C_l = 1.135$ ,  $C_d = 0.0055$ ,  $C_l/C_d = 206.24$ ) is excellent, the design must adhere to the new camber constraint. For future iterations, the design should target a camber within the 1-6% range. Additionally, while maximizing the L/D ratio, the pitching moment ( $CM = -0.272$ ) should also be optimized, potentially by adjusting the camber location or thickness distribution to achieve a more favorable CM.", "Valid": false



With this feedback, the Design Engineer agent revises the design to include improvements suggested. The output below shows the reasoning process followed.

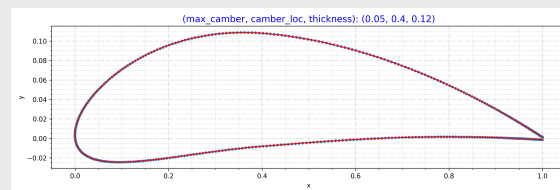
### Design revision 2: Designer's reasoning

**Design Parameters:** - Maximum Camber: 0.05 (5%) - Camber Location: 0.4 (40% chord) - Thickness: 0.12 (12%)

**Reasoning for Design Modification:** - The maximum camber was reduced to 5% to comply with the updated requirement of camber being between 1% and 6%. - The camber location was set at 40% of the chord to potentially improve lift and overall airfoil performance, as moving the camber location towards the leading edge increases lift. - The thickness was maintained at 12% to manage drag effectively at Mach 0.6 while avoiding excessive drag coefficients.

**Knowledge Source Reference:** - Moving the camber location towards the leading edge increases lift. - A more aft maximum camber location improves airfoil performance. - Thickness should be within 6% to 20% to manage drag at Mach 0.6.

The revised airfoil design, identified as Revision 2, has been generated and saved. This design adheres to the updated constraints and aims to maximize the lift-to-drag ratio while optimizing the pitching moment."



The Systems Engineer feedback on Design Revision 2 is as follows:

### Design revision 2: Systems Engineer feedback

"The airfoil profile, identified as a NACA 5412 based on the geometric parameters (camber: 0.05, camber location: 0.4, thickness: 0.12), exhibits a well-defined shape with a rounded leading edge and sharp trailing edge, typical for a lifting airfoil. The aerodynamic performance is excellent, with a high lift coefficient ( $C_l$ : 0.582) at 0 degrees angle of attack and a remarkably low drag coefficient ( $C_d$ : 0.00578), resulting in an outstanding lift-to-drag ratio of 100.71. This design adheres to all specified functional requirements, including the NACA 4-digit family, geometric parameter constraints, and operating conditions. The  $C_l$  value is above 0.5, making the design valid by the rule of thumb. For further improvement, it is crucial to verify the critical Mach number to ensure it is greater than 0.6 at the operating Mach number of 0.6 to mitigate compressibility effects. Additionally, a detailed pressure distribution analysis and robustness study at slightly off-design conditions would enhance the design's confidence." "Valid": True

The Manager intervenes again here to suggest design modification in order to improve the lift coefficient further.

### Design revision 2: Manager feedback

"Can you suggest improvement to make Cl close to 0.7? Please mark this design as invalid"

The Systems Engineer agent then updates its feedback for the Design Engineer to include the Manager's comments for improving Cl values further

### Design revision 2: Systems Engineer updated feedback

"The current airfoil (NACA 5412) provides a lift coefficient (Cl) of 0.582, which is good, *but the user has requested an improvement to achieve a Cl closer to 0.7*. To increase the lift coefficient at 0 degrees angle of attack, it is recommended to increase the maximum camber of the airfoil. For instance, exploring NACA 6412 (camber: 0.06, camber location: 0.4, thickness: 0.12) or NACA 7412 (camber: 0.07, camber location: 0.4, thickness: 0.12) could help achieve the target Cl while remaining within the NACA 4-digit series constraints. This adjustment would likely increase Cl, but further analysis would be needed to ensure the lift-to-drag ratio remains optimized and the critical Mach number requirement is still met. Valid: False"

The Design Engineer acts on this feedback to update the design further to meet the new request from the Manager

### Design revision 3: Designer's reasoning

"To address the feedback and requirements, I need to determine the best design parameters for a revised airfoil that meets the specified criteria. The goal is to achieve a lift coefficient (Cl) closer to 0.7 while maximizing the lift-to-drag ratio (Cl/Cd) at the specified operating conditions. The feedback suggests exploring NACA 6412 and NACA 7412, which have higher camber values than the current NACA 5412.

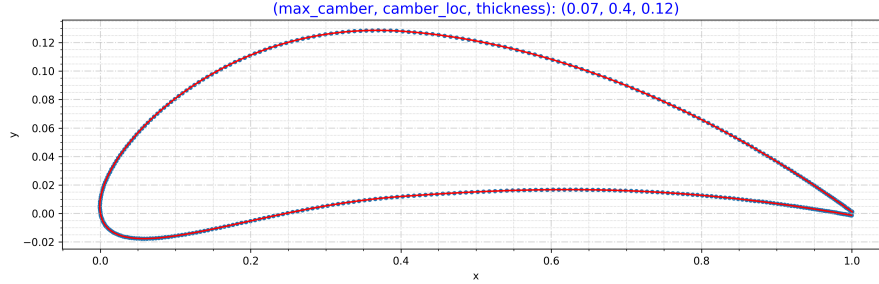
The requirements specify: - **Maximum Camber (m)**: 0% to 9% - **Position of Maximum Camber (p)**: 0% to 90% - **Maximum Thickness (t)**: 6% to 20%

Given the feedback and requirements, I will consider the following: - Increasing the maximum camber to improve the lift coefficient. - Keeping the camber location at 0.4 to maintain a balance between lift and drag. - Maintaining the thickness at 0.12 to avoid excessive drag.

I will generate a revised airfoil design using the 'generate airfoil' tool with the following parameters: - Maximum Camber: 0.07 (for NACA 7412) - Camber Location: 0.4 - Thickness: 0.12"

The iterative review and design process, spanning Stages 4 and 5, continues until a satisfactory design candidate is selected by the Manager. This loop, involving the Systems Engineer and Manager, ensures that the design not only met the technical requirements

but also aligned with broader human user’s goals and considerations. The final design candidate proposed by the Design Engineer is shown in Figure 7.6.

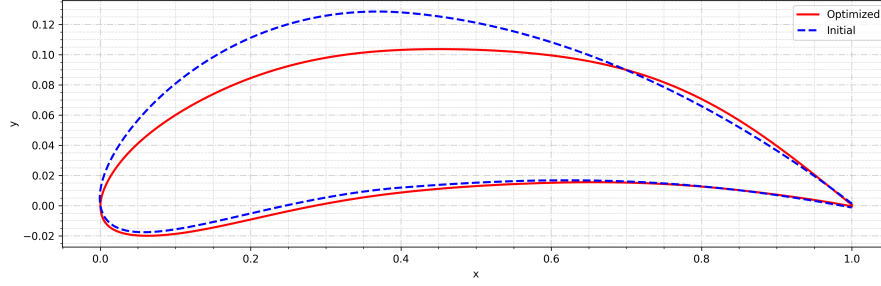


**Figure 7.6:** *Final candidate design chosen for the optimization process after the iterative design phase. Note that the camber value for this design exceeds the 6% criteria defined by the Manager at design iteration 1. However, due to the new request from the Manager to improve  $C_l$  values further, the agents identify the need to increase the camber values beyond this range to meet the competing requirements of higher lift and pitching moment.*

## Design Optimization

Stage 6 marks the final phase: design optimization. The design selected from the iterative review process serves as the starting point for this optimization effort. The optimization objective is defined as maximizing the lift-to-drag ratio ( $L/D$ ) at a Mach number ( $Ma$ ) of 0.6, Reynolds number ( $Re$ ) of  $5 \times 10^6$ , and angle of attack ( $AoA$ ) of 0 degrees. This objective function directly reflects a critical performance metric for airfoil designs – maximizing lift while minimizing drag. The design variables are the coordinates of the airfoil profile. For this phase, the Design Engineer utilizes the Optimization tool available inside NeuralFoil and AeroSandbox, where two default constraints for minimum local thickness and trailing edge values are enforced to ensure convergence to a feasible solution. Figure 7.7 shows a comparison between the initial design and the optimized design obtained at the end of this stage.

Once an optimized design is obtained, the design process concludes at this point. Further iterations may be necessary to refine the design for final product realization, potentially involving structural and CFD analyses, which is considered beyond the scope of this present study.



**Figure 7.7:** Comparison between optimized design and initial starting design selected through the iterative design process. The optimized design has performance values  $C_l = 0.811$ , and  $C_d = 0.0044$  against the initial starting design with  $C_l = 0.8077$  and  $C_d = 0.0061$ . Note that since the optimization process uses an initial design that satisfies (to a large extent) both functional and non-functional requirements, the end design does not suffer from qualitative shortcomings such as thin trailing edge which may lead to other challenges in manufacturing and robustness.

## 7.6 Limitations and Future Works

The presented framework, while demonstrating the potential for integrating multi-agent systems (MAS) into iterative engineering design processes, contains certain limitations. Our approach currently embodies an academic exploration of this integration, focusing on a simplified representation of engineering tasks. Engineering design, by its inherent nature, frequently involves intricate and nuanced decision-making processes that extend beyond the scope of our current implementation. While the agents exhibited adaptability in navigating decision-making scenarios with the help of the knowledge graph, maintaining consistency across sophisticated engineering tasks remains a significant challenge. The problem formulation used in our work shows a relatively straightforward pathway, which is often a simplification compared to the complexities encountered in real-world engineering design. Unexpected sub-processes and iterations are common occurrences during the design cycle, deviating from the linear progression assumed in our demonstration.

Furthermore, the tools leveraged by the Design Engineer agent are presently implemented as simple, Python-based modules, facilitating ease of integration within the workflow. However, a major impediment to broader adoption lies in the reliance on proprietary software crucial to engineering practice. Seamless integration with industry-standard Computer-Aided Design (CAD) and Computer-Aided Engineering (CAE) tools presents

a considerable hurdle for our MAS framework. The complexity and closed nature of these tools often preclude direct interaction with agents, necessitating the development of intermediary layers and custom interfaces, a task that demands substantial resources and expertise. Future work will focus on addressing these limitations, exploring more complex workflows that incorporate agents to accomplish Computational Fluid Dynamics (CFD) analyses on final design.

In the future, a key area of development will be the agents' ability to dynamically create and manage tools within the design workflow. Currently, our agents operate within a pre-defined toolset. The ability to generate tools on demand – for example, custom scripts for data analysis or specialized modules for simulation – would significantly enhance the framework's flexibility and utility. This dynamic tool creation capability represents a substantial challenge, requiring advancements in automated code generation and agent reasoning, but is important to enable more autonomous and adaptive engineering design processes. We believe this capability will be pivotal in realizing the full potential of MAS within the engineering domain.

## 7.7 Summary

In this work, we propose a novel multi-agent framework that leverages large language models (LLMs) to facilitate and accelerate the solution for a pedagogical challenge of designing high-performance NACA airfoils. We focus on the design of 4-digit NACA airfoils, for demonstration purposes. This framework comprises three specialized AI agents, each playing a distinct role in the iterative design cycle. The core agents are: a Graph Ontologist, a Design Engineer, and a Systems Engineer. The Graph Ontologist utilizes a prompt engineering approach to curate specialized knowledge graphs by extracting relevant information related to airfoil design and development from scientific literature. The extracted data is structured and organized within the graph, enabling efficient retrieval and utilization by other agents. Both Design and Systems Engineer are provided with

their own customized knowledge graph that is suited for their roles and expertise. The Design Engineer is responsible for generating and modifying airfoil designs based on the information contained within the knowledge graph in consultation with the Systems Engineer agent. The Systems Engineer reviews designs generated by the Design Engineer, provides feedback for improvement, and ensures alignment with technical requirements. This agent also receives input from a human Manager to translate user decisions into actionable design feedback.

The Design and Systems Engineer agents accomplish the design process through an iterative process. The Design Engineer, guided by its knowledge graph and design tools, produces an initial set of airfoil designs. The Systems Engineer then assesses these designs, evaluating both their quantitative performance characteristics (e.g., lift and drag coefficients) and qualitative attributes (e.g., manufacturability, robustness) as defined in the technical specification. This feedback is translated into actionable design improvement strategies, such as adjustments to airfoil geometry. The agents access and utilize their individual knowledge graphs, ensuring consistency and facilitating knowledge transfer between design iterations. To ensure that the iterative process converges on a suitable design, a human Manager is included in the final design selection phase. The Manager determines whether a design is valid based on his requirements. This process is repeated until a design that meets the Manager's requirements is obtained.

This synergistic approach prioritizes both quantitative and qualitative requirements during design iteration, ensuring that the final design meets overall acceptance criteria. In future work, we will investigate the use of additional agents such as a CFD analyst to help address next steps in the design sequence along with methods to generate simple design tools dynamically during the process. In summary, the framework presented here demonstrates a path toward automating and augmenting the engineering design process, potentially improving design quality and accelerating innovation.

# CHAPTER 8

## Agentic Risk-Aware Set-Based Engineering Design

### 8.1 Introduction

Engineering design is a profoundly complex and multifaceted process, characterized by a non-linear workflow and frequent iterative cycles. The conceptual design phase, in particular, represents the most critical juncture in product development, where decisions can commit up to 75% of the total product lifecycle cost [285]. During this phase, designers must navigate a vast and often poorly defined design space to identify promising solutions. This initial exploration is inherently iterative, as a wide array of alternatives must be considered and progressively refined. The conventional approach to this challenge, often termed point-based design, involves an early commitment to a single design concept, which is then subjected to successive refinements and optimizations. While straightforward, this method introduces significant inflexibility and risk. Committing to a single solution when the problem is ill-defined can lead to design fixation, a well-documented cognitive bias that impedes the exploration of alternative, and potentially superior, solutions [286]. The cascading effects of such an early decision can result in costly, late-stage redesigns when unforeseen constraints or better alternatives emerge.

To address the profound limitations of point-based methods, the Set-Based Design (SBD) paradigm, which has its roots in the highly successful Toyota Production System, offers a more robust and flexible alternative [287]. Rather than selecting a single point solution and refining it, SBD advocates for the simultaneous exploration of broad sets of design alternatives. The design team works concurrently with multiple potential solutions, maintaining ambiguity and delaying critical decisions until more information is available. This process involves gradually narrowing the design space by eliminating entire regions that are proven to be infeasible or non-promising, thereby mitigating the risk of premature commitment and fostering a more comprehensive exploration of potential solutions [288]. Mathematically, this convergent process can be represented as a sequence of operations on sets of design parameters. If we define the initial, continuous design space as  $\mathcal{D} \subseteq \mathbb{R}^n$ , the SBD process can be modeled as a sequence of discrete sets  $S_k$  at iteration  $k$ , such that  $\mathcal{D} \supseteq S_0 \supseteq S_1 \supseteq \dots \supseteq S_f$ , where  $S_f$  is the final set of selected designs. Each step in this sequence is governed by the relation  $S_{k+1} = \mathcal{F}_k(\mathcal{M}_k(S_k))$ , where  $\mathcal{M}_k$  is a mapping operator representing design evaluation or modification, and  $\mathcal{F}_k$  is a filtering operator that narrows the set based on performance, feasibility, or risk criteria.

An effective and intelligent design filtering strategy is the cornerstone of the SBD methodology. One powerful approach for implementing the filtering operator  $\mathcal{F}_k$  is risk-based design selection. In this context, risk is formally defined as a function of both the probability and the consequence of failure to meet a performance objective [289], and it explicitly accounts for various sources of uncertainty. These uncertainties are typically categorized as either aleatory (inherent randomness in the system or environment) or epistemic (a reducible lack of knowledge or inaccuracy in predictive models) [290]. By evaluating design candidates based on their expected risk profile, this method ensures that the selected designs are robust to variations in operating conditions and model-form uncertainty. The principles of incorporating risk into design decisions are well-established, drawing from decision theory and robust design methodologies [291, 292]. To manage computational costs in early-stage design exploration, performance evaluations typically

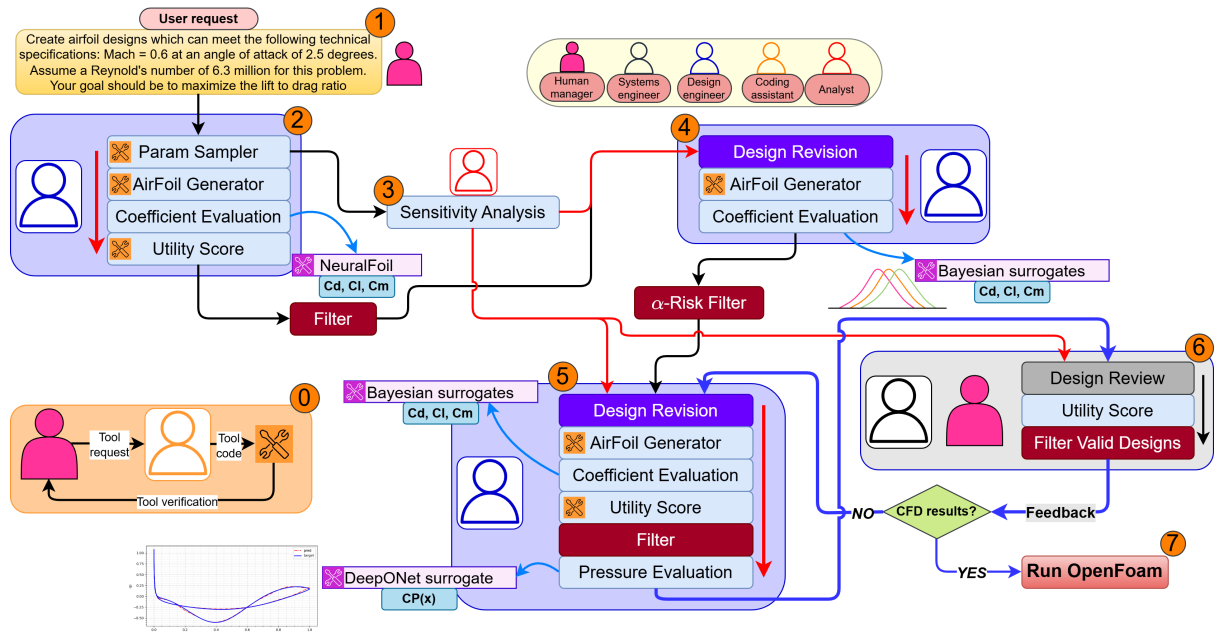


rely on simplified, low-fidelity physical models (e.g., potential flow codes in aerodynamics) [293]. While these models offer rapid assessments, their inherent simplifications and missing physics—such as neglecting viscous effects or turbulence—can lead to grossly inaccurate performance predictions, potentially causing superior designs to be prematurely discarded.

To bridge this gap between computational cost and predictive accuracy, surrogate models, or metamodels, have emerged as an indispensable tool in modern engineering design [294]. Among the various types of surrogates, those based on neural networks (NNs) have shown significant promise in recent years for their ability to approximate highly non-linear, high-dimensional functions. These models are trained on a limited set of high-fidelity data points and learn the complex mapping between design parameters and performance outcomes. Their primary advantage is a significant shift in computational timescale, reducing evaluation times from hours or days for a single high-fidelity simulation to milliseconds. This speed enables the exhaustive evaluation of thousands of design candidates, making large-scale SBD exploration computationally feasible. Furthermore, unlike simplified physics models, well-trained NNs can capture the intricate physical phenomena present in the high-fidelity data, leading to far more reliable evaluations within the SBD filtering process.

This paper addresses the existing research gap in the integration and intelligent orchestration of these advanced design paradigms. While SBD, risk analysis, and surrogates are powerful individually, their synergistic combination into a cohesive, semi-autonomous workflow remains a significant challenge. We introduce a novel framework that integrates these concepts for the conceptual design of airfoils, orchestrated by an innovative multi-agent system driven by Large Language Models (LLMs). This framework leverages the reasoning and coordination capabilities of LLMs to manage the iterative design loop of evaluation, filtering, and review. The primary contributions of this work are threefold: (1) the development and application of a comprehensive set-based design methodology for airfoil design, featuring a robust, risk-based filtering mechanism; (2) the implementation of a multi-stage design evaluation and refinement process powered by pre-trained neural

network surrogates for high-speed, high-fidelity assessment; and (3) the introduction of a novel LLM-driven multi-agent framework for automating design evaluation and ranking, augmented by a human-in-the-loop feedback system for final design validation. The remainder of this paper details the architecture of this framework, demonstrates its application to an airfoil design case study, and discusses the results and implications for future intelligent design systems. The complete workflow for this framework is shown in Figure 8.1.



**Figure 8.1:** Schematic for LLM-based set-based design workflow with risk filtering strategy. The LLM workflow consists of four agents: Design Engineer, Systems Engineer, Coding Assistant, and Analyst, each assigned a designated role and task. The workflow utilizes different tools that are developed by the Coding Assistant following human manager's instructions. The workflow starts with the human manager providing a design problem to the Design Engineer. The Design Engineer, equipped with a set of engineering tools, then proceeds to developing, analyzing, and filtering design candidates based on certain rules. After a sequence of design filtering and improvements, an iterative design review and improvement cycle commences with the Systems Engineer reviewing and rating the design solutions based on a utility score, in comparison with a benchmark airfoil design. Final design candidates are reviewed jointly by the Systems Engineer and human manager, who determines which designs to retain in the design process. The process ends once the human manager determines a set of viable candidate designs and requests for CFD analysis of these candidates.

## 8.2 Problem statement

The design of an airfoil profile is a classic engineering optimization challenge, involving a delicate trade-off between maximizing lift and minimizing drag under specific flight conditions. Key aerodynamic metrics, such as the coefficients of lift ( $C_L$ ), drag ( $C_D$ ), and pitching moment ( $C_M$ ), are highly sensitive to the airfoil's geometry and the flow regime, which is characterized by the Reynolds number (Re), Mach number (Ma), and angle of attack (AoA). The primary goal of airfoil design problem is to identify a geometry that yields superior performance for a given set of operating conditions.

To facilitate an automated design process, the airfoil geometry must be described by a finite set of parameters that can be manipulated during iterative design process. A widely adopted method for this purpose is the Class-Shape Transformation (CST) parameterization, introduced by Kulfan [284]. Unlike other airfoil parameterization methods such as NACA, the CST method does not require well-defined equations for representing airfoil profile. It also provides a robust and intuitive way to represent a wide range of airfoil shapes with a relatively small number of design variables. The vertical coordinates of the upper ( $y_u$ ) and lower ( $y_l$ ) airfoil surfaces are defined as a product of a class function,  $C(x)$ , and a shape function,  $S(x)$ , plus a term for a finite trailing edge thickness,  $y_{TE}$ . The formulation in a normalized coordinate system ( $x \in [0, 1]$ ) is given by:

$$y(x) = C(x)S(x) + x \cdot y_{TE} . \quad (8.2.1)$$

The class function,  $C_{N_2}^{N_1}(x) = x^{N_1}(1 - x)^{N_2}$ , defines the fundamental topology of the airfoil class. For conventional airfoils, the exponents  $N_1 = 0.5$  and  $N_2 = 1.0$  are chosen to ensure a rounded leading edge and a sharp, finite-angle trailing edge, respectively. The shape function is a linear combination of Bernstein polynomials, which provides the detailed contour:

$$S(x) = \sum_{i=0}^n w_i B_{i,n}(x) = \sum_{i=0}^n w_i \binom{n}{i} x^i (1-x)^{n-i}. \quad (8.2.2)$$

where  $n$  is the number of weights defining the airfoil surface. In this formulation, the weights,  $w_i$ , associated with each Bernstein polynomial,  $B_{i,n}(x)$ , serve as the design variables that control the airfoil’s final shape. A distinct set of weights is used for the upper and lower surfaces, providing comprehensive control over the airfoil’s camber and thickness distribution. In this study, we adopt the range of CST parameters as defined in [10] which allows us to use this existing CFD data for training our neural surrogates.

This study frames the challenge as a single-point design problem. The primary objective is to design an efficient airfoil that maximizes the lift coefficient ( $C_L$ ) under specific flow conditions. The chosen operating point is defined by a Reynolds number of  $\text{Re} = 6.3$  million, a freestream Mach number of  $\text{Ma} = 0.6$ , and an angle of attack of  $\text{AoA} = 2.5$  degrees. These conditions were specifically selected because they correspond to the well-documented experimental test case for the RAE2822 airfoil [6]. The availability of extensive experimental and computational data for this case provides a crucial benchmark for validating the simulation results and assessing the performance of the final design generated by our LLM-agent framework. While the primary objective is lift maximization, other critical performance metrics, including the drag coefficient ( $C_D$ ), the moment coefficient ( $C_M$ ), and the surface pressure distribution ( $C_P$ ), will be evaluated for airfoil designs to ensure a well-rounded and aerodynamically efficient design.

### 8.3 Agentic Design as Sequential Decision-Making Under Uncertainty

In this section, we present a theoretical interpretation of the proposed multi-agent design framework, formalizing the agentic workflow as a sequential decision-making process under uncertainty. Specifically, we cast the methodology as an iterative procedure over sets

of candidate designs, governed by stochastic evaluation operators and risk-aware filtering mechanisms. This perspective provides a mathematical foundation for understanding the roles of individual Large Language Model (LLM) agents, the treatment of epistemic and aleatory uncertainties, and the incorporation of quantitative risk metrics to guide the efficient exploration of high-dimensional engineering design spaces.

We adopt the set-based design (SBD) paradigm [155], which shifts the focus from optimizing a single point to progressively narrowing a space of possibilities. Let  $\mathcal{D} \subset \mathbb{R}^n$  denote the continuous design space parameterized by  $n$  variables, such as the Class Shape Transformation (CST) coefficients used to define aerodynamic profiles. The SBD process can be mathematically represented as a monotonically decreasing sequence of subsets:

$$\mathcal{D} \supseteq S_0 \supseteq S_1 \supseteq \cdots \supseteq S_K, \quad (8.3.1)$$

where  $S_k$  denotes the active set of candidate designs at the  $k$ -th iteration. The evolution of this candidate set is driven by the application of two distinct, sequentially applied operators. Formally, the transition from  $S_k$  to  $S_{k+1}$  is expressed as:

$$S_{k+1} = \mathcal{F}_k(\mathcal{M}_k(S_k)), \quad (8.3.2)$$

where  $\mathcal{M}_k : S_k \rightarrow \tilde{S}_k$  represents a modification operator responsible for the generation and refinement of candidate geometries, effectively exploring the local design space. Conversely,  $\mathcal{F}_k : \tilde{S}_k \rightarrow S_{k+1}$  acts as a filtering operator that discards inferior designs based on performance and risk criteria. Within our framework, these abstract operators are instantiated by specialized LLM agents: the Design Agent executes the modification mapping  $\mathcal{M}_k$ , while the Systems Engineering Agent, guided by risk-aware filters, implements the evaluation mapping  $\mathcal{F}_k$ .

A critical aspect of early-stage engineering design is the presence of both epistemic uncertainty (e.g., surrogate model inaccuracies) and aleatory uncertainty (e.g., variable

operating conditions) [295]. Consequently, each design vector  $x \in \mathcal{D}$  is evaluated through a stochastic performance model. We denote the performance metrics of a design, such as the coefficients of lift ( $C_L$ ), drag ( $C_D$ ), and moment ( $C_M$ ), as a random vector  $Y(x)$  distributed according to a probability density function  $p(y \mid x)$ . This distribution is typically induced by the underlying predictive tools, such as Bayesian neural networks or other probabilistic surrogate models. To aggregate these multivariate metrics into a unified evaluation criterion, we define a scalar utility functional [296]:

$$U(x) = \mathbb{E}[u(Y(x))], \quad (8.3.3)$$

where the function  $u(\cdot)$  maps the stochastic performance vector to a deterministic utility score, reflecting the human Manager's design objectives.

Basing decisions solely on expected utility can leave the design vulnerable to critical failures, particularly when the performance distribution exhibits heavy tails [157]. To ensure robustness under uncertainty, the filtering operator  $\mathcal{F}_k$  explicitly incorporates formal risk measures. For a given performance metric of interest  $X$  (e.g.,  $C_L$ ), we utilize the Conditional Value-at-Risk (CVaR) [159] at a specified confidence level  $\alpha \in (0, 1)$ . The CVaR represents the expected value of the worst-case outcomes and is defined as:

$$\text{CVaR}_\alpha(X) = \mathbb{E}[X \mid X \geq \text{VaR}_\alpha(X)], \quad (8.3.4)$$

where  $\text{VaR}_\alpha$  is the corresponding  $\alpha$ -quantile of the distribution. The filtering operator is subsequently defined as an indicator function:

$$\mathcal{F}_k(x) = \begin{cases} 1, & \text{if } \begin{cases} U(x) \geq \tau^*, & k \in \mathcal{K}_U \\ \text{CVaR}_\alpha(X) \geq \gamma^*, & k \in \mathcal{K}_R \end{cases} \\ 0, & \text{otherwise} \end{cases} \quad (8.3.5)$$

where  $\tau^*$  and  $\gamma^*$  are the utility and risk thresholds respectively,  $\mathcal{K}_U$  and  $\mathcal{K}_R$  represent

the iteration using either utility function or CVaR as a risk-filtering strategy. This dual-constraint formulation guarantees that the retained designs are not only high-performing in expectation but also robust against adverse uncertainty realizations.

In this framework, the state at step  $k$  is defined by the current set of candidates,  $s_k = S_k$ . The action  $a_k = \mathcal{M}_k$  corresponds to the specific modification strategy or heuristics applied to  $s_k$ . The state transition is deterministically governed by the composite application of modification and filtering:

$$s_{k+1} = \mathcal{F}_k(\mathcal{M}_k(s_k)). \quad (8.3.6)$$

Assuming mild regularity conditions on the filtering operator, the sequence of subsets  $\{S_k\}$  forms a monotonically shrinking space, enforcing  $S_{k+1} \subseteq S_k$ . Eventually, the agentic process converges to a terminal set of designs  $S_K$  that satisfies the strict criteria:

$$S_K = \{x \in \mathcal{D} \mid U(x) \geq \tau^*, \text{CVaR}_\alpha(X) \geq \gamma^*\}. \quad (8.3.7)$$

Ultimately, this theoretical formulation underscores two fundamental properties of the proposed multi-agent framework: it naturally balances exploration and exploitation through the alternation of the modification ( $\mathcal{M}_k$ ) and filtering ( $\mathcal{F}_k$ ) operators and it enforces structural robustness via CVaR-based risk constraints. These distinct characteristics elevate the proposed agentic framework beyond classical point-based optimization routines, establishing a basis for its application in complex, uncertain engineering design workflows and directly motivating the multi-agent architecture described in the subsequent sections.

## 8.4 Methodology

In this section, we discuss the key elements and processes associated with our MAS framework that orchestrates the SBD paradigm.

### 8.4.1 Agent: Coding Assistant

A central challenge in developing robust multi-agent systems for engineering design is ensuring the agents have access to reliable tools necessary in the design workflow. To address this, our framework incorporates a specialized ‘Coding Assistant’ powered by Gemini-2.5-pro. This agent acts as a dedicated tool-smith, translating high-level, natural language requests from a human user into executable code. This process is intentionally designed as an offline, human-in-the-loop workflow, separate from the main autonomous design loop. In this setting, the user specifies a required function: for example, a tool for the ‘Design Engineer’ agent to generate design candidates using Latin Hypercube Sampling and the ‘Coding Assistant’ produces the corresponding code. The human manager then evaluates this code for correctness and efficiency, enabling an iterative feedback loop for refinement until the tool is fully validated. By pre-defining and sanctioning the entire toolset before the primary design task begins, we mitigate the risks of on-the-fly code generation, such as logical errors or redundant functions. This deliberate separation of concerns supervised tool creation versus autonomous tool application is a critical architectural choice that provides the core design agents with a foundation of robust tools, thereby enhancing the overall reliability and predictability of the multi-agent system. An example prompt provided to the Coding Assistant is shown in figure 8.2.

### 8.4.2 Agent: Design Engineer

The Design Engineer agent serves as the primary generative and iterative agent within the multi-agent framework, embodying the role of a traditional design specialist. Its core responsibility is to navigate the design space through a two-phase process: an initial, divergent exploration followed by a convergent, data-driven refinement. In the initial phase, the agent autonomously generates a diverse population of candidate airfoil geometries by sampling the 9-dimensional design space defined by the CST parameters. It then rapidly estimates key aerodynamic performance metrics, such as the coefficients of lift



### User input for creating tool: Param sampler

```
" I want to create a tool which enables the following:
  • Create samples from the design space for the 9 CST parameters given by
    <designspace>
    "CST1": (0.0644, 0.1932),
    "CST2": (0.0688, 0.2064),
    "CST3": (0.0961, 0.2883),
    "CST4": (0.0961, 0.2882),
    "CST5": (0.1010, 0.3030),
    "CST6": (0.0680, 0.2039),
    "CST7": (0.1126, 0.3377),
    "CST8": (0.0381, 0.1143),
    "CST9": (-0.0586, -0.0195)
    </designspace>
  • Enable user to choose from different sampling strategies such as Latin Hypercube, Sobol, random etc by
    enabling an input argument to the tool. You can use qmc library from scipy for this.
  • Provide an input argument for number of samples to be generated to the tool.
  • Provide input argument to the tool for 3 scalar flow parameters: Ma (float), AoA (float), Re (float). These
    parameters remain the same for all samples in the design space.
  • Create an array named 'nninputs' with size: numsamples x 12, where the first three columns on the array
    contain the 3 flow parameter and the remaining 9 columns contain the CST parameters sampled from the
    design space.
  • Create a list of 'designid' such as ID-1, ID-2,....., ID-numsamples for the number of design samples.
  • Convert array 'nninputs' to a list. Then export the following dictionary as a pickle object.
    <exportdict>
      'samples': nninputs (list)
      'designid': designid
    </exportdict>
```

**Figure 8.2:** Sample instruction provided to the Coding Agent by the human user to create a parameter sampling tool based on specific process requirements.

( $C_L$ ) and drag ( $C_D$ ), using computationally inexpensive surrogate models to assign a utility score and filters a set of promising candidates. During the later convergent stages, the agent's role shifts to focused improvement, operating in a collaborative feedback loop with the Systems Engineer agent and potentially a human user. To facilitate intelligent design modifications, the Design Engineer relies on quantitative sensitivity analysis results provided by the Analyst agent, which correlate the CST parameters to aerodynamic performance. Armed with this information, the agent can make informed, non-random adjustments, simultaneously modifying an entire set of promising designs to efficiently steer their performance toward a desired optimum based on system-level feedback, thereby accelerating convergence to a high-quality solution.

### 8.4.3 Agent: Systems Engineer

The Systems Engineer agent functions as the primary design evaluator and strategic guide within the multi-agent framework, responsible for providing assessment of airfoil designs generated by the Design Engineer. This agent uses Gemini-2.5-pro multi-modal Large Language Model, which enables a holistic analysis that mirrors the expert judgment of a human engineer. It simultaneously processes quantitative performance coefficients - specifically, the coefficients of drag ( $C_D$ ), lift ( $C_L$ ), and moment ( $C_M$ ), alongside qualitative graphical data, such as surface pressure distribution curves and airfoil profiles, to identify promising design candidates. The agent operates in two distinct modes to adapt to different stages of the design workflow. In its fully autonomous mode, typically employed during the broad exploration phase, it systematically analyzes a large number of designs. It assigns a utility rating to each candidate based on predefined performance metrics and evaluates the quality of the pressure distribution by comparing it against an established benchmark, such as the RAE2822 airfoil, to autonomously filter for valid designs without human intervention. As the design process converges and the set of candidate designs is reduced to a manageable number where human involvement is pragmatic, the agent transitions to a semi-autonomous mode. In this configuration, it functions as a collaborative partner to a human manager, where both parties review and rate the final candidates in parallel. Crucially, beyond mere evaluation, the Systems Engineer agent closes the iterative design loop by providing actionable feedback. Based on its comprehensive assessment, and upon incorporating specific guidance from the human manager, it formulates and communicates specific design improvement strategies to the Design Engineer agent, thereby guiding the targeted refinement of the airfoil geometries.

### 8.4.4 Agent: Analyst

The Analyst serves as the primary interpretive agent within the framework, tasked with translating raw numerical data into actionable design intelligence for the Design Engineer

agent. Its core function is to address a fundamental challenge associated with abstract geometric parameterizations like the Class-Shape Transformation (CST) method. While CST provides a powerful and flexible means to represent airfoils, its weight parameters ( $w_i$ ) lack a direct, intuitive connection to classical aerodynamic shape characteristics such as maximum thickness, camber, or leading-edge radius. This decoupling makes it difficult for a designer, human or artificial, to predict how a change in a specific CST weight will affect aerodynamic performance. The Analyst agent bridges this knowledge gap by performing a quantitative sensitivity analysis, which systematically evaluates the statistical correlations between each of the CST design variables and the key performance metrics ( $C_L$ ,  $C_D$ , etc.). By processing the results of numerous simulations, this agent computes a sensitivity map, often in the form of a correlation matrix, which quantifies the magnitude and direction of influence each parameter has on the design objectives. Crucially, the Analyst does not merely pass this raw data onward; its primary contribution is the synthesis of these findings into a concise set of heuristic rules and strategic recommendations. For example, it might conclude that “increasing the third upper surface weight,  $w_{u,3}$ , has a strong positive correlation with the lift coefficient ( $C_L$ ) but a weak correlation with the drag coefficient ( $C_D$ ).” This distilled guidance empowers the Design Engineer agent to make informed, non-random modifications, effectively steering the iterative design process toward optimal regions of the design space with greater efficiency and purpose.

#### 8.4.5 Tools

Tools form an important component of our MAS framework, and provide our agents with the ability to execute tasks. We provide a description of all major tools including filtering strategies adopted in this section. Note that all these tools are developed by the Coding assistant on request by the human manager prior to commencement of the main design loop.

## Param Sampler

The initial exploration of the design space is orchestrated by a dedicated utility, the ‘Param Sampler’ tool, which is responsible for generating the population of candidate airfoil designs. This tool operates within a 9-dimensional design space defined by the weight parameters of the Class-Shape Transformation (CST) method. The bounds for each of the nine CST variables, which collectively describe the airfoil geometry, are precisely constrained to the ranges specified in [10]. A key feature of the sampler is its methodological flexibility, enabling the selection of various sampling strategies via an input argument. In addition to standard pseudo-random sampling, the tool integrates Quasi-Monte Carlo (QMC) methods such as Latin Hypercube and Sobol sequences, implemented using the ‘scipy.stats.qmc’ library (this was done for consistency of Analyst response). The use of these advanced techniques is critical for ensuring a more uniform and low-discrepancy distribution of sample points across the high-dimensional parameter space, which facilitates a more efficient and comprehensive exploration compared to simple random sampling, especially when the number of evaluations is limited [297]. Operationally, the tool requires the user to specify the total number of samples to generate, as well as three scalar flow parameters: Mach number ( $Ma$ ), angle of attack ( $AoA$ ), and Reynolds number ( $Re$ ), which are extracted from the initial design request and remain constant for all generated airfoil geometries. The resulting output is a structured set of design candidates, each defined by a unique vector of nine CST parameters, ready for subsequent aerodynamic evaluation under a consistent flight regime.

## Airfoil Generator

The ‘Airfoil Generator’ tool helps in converting the abstract parametric design space into concrete airfoil geometry coordinates required for aerodynamic evaluation. This tool ingests a vector of CST weights and generates a set of cartesian coordinates that define the airfoil’s profile. This transformation is governed by the CST formulation shown in

equation 8.2.2, where the vertical coordinates of the upper ( $y_u$ ) and lower ( $y_l$ ) surfaces are expressed as the product of a Class function,  $C(x)$ , and a Shape function,  $S(x)$ . Note that the first weight chosen on  $y_l$  is equal to the first parameter of  $y_u$  to maintain  $\mathcal{C}^2$  continuity of the leading edge. The resulting point clouds for the upper and lower surfaces are then assembled into a single, ordered array suitable for other steps in the framework, typically by traversing the upper surface from the trailing edge to the leading edge and then back along the lower surface.

## Coefficient Evaluation

For evaluating aerodynamic performance for different designs, two different surrogate models are used. These surrogates are developed to predict aerodynamic coefficients  $C_D$ ,  $C_L$ , and  $C_M$  across a range of Ma, Re, and AoA.

- *NeuralFoil*: For the rapid initial screening of airfoil candidates, the Design Engineer agent utilizes NeuralFoil, a deep learning-based surrogate model developed to provide near-instantaneous predictions of aerodynamic coefficients [276]. NeuralFoil approximates the solution to the inviscid Euler equations, employing a neural network architecture to map a given airfoil geometry, defined by its coordinates or parametric representation along with the angle of attack (AoA) and Mach number (Ma) to the corresponding coefficients of lift ( $C_L$ ), drag ( $C_D$ ), and pitching moment ( $C_M$ ). The validated operating range for NeuralFoil covers subsonic and transonic flow conditions, specifically for Mach numbers between 0 and 0.75 and angles of attack from -5 to 15 degrees. A critical limitation of this framework, however, is its foundation in inviscid flow theory. As such, NeuralFoil does not model viscous effects, such as boundary layer development, skin friction drag, or flow separation. Consequently, its drag predictions primarily account for wave drag and are not representative of the total drag experienced by the airfoil, and its lift predictions become unreliable near stall conditions.

- *Bayesian surrogate*: To facilitate a robust, risk-informed filtering process, a probabilistic modeling approach is adopted to quantify the uncertainty associated with aerodynamic performance predictions. This is achieved by developing a Bayesian Neural Network (BNN) designed to learn the complex mapping from the airfoil’s geometric and operational parameters to its aerodynamic coefficients. The input vector to the model consists of the nine Class-Shape Transformation (CST) parameters defining the airfoil geometry, along with the Reynolds number (Re), Mach number (Ma), and angle of attack (AoA). Since associated uncertainties may differ for each performance metric, separate and independent BNNs are trained to predict the coefficients of drag ( $C_D$ ), lift ( $C_L$ ), and moment ( $C_M$ ). The primary purpose of these networks is not to provide a single point estimate but to generate a full predictive distribution for each performance coefficient. This probabilistic output is used during the alpha-risk filtering stage, where the epistemic uncertainty of surrogate models is considered for design filtering. For details related to the Bayesian surrogate, please refer to Appendix A4.2.

## Pressure Evaluation

While integral aerodynamic coefficients such as lift and drag provide a primary measure of airfoil performance, a detailed analysis of the surface pressure distribution, characterized by the pressure coefficient ( $C_P$ ) is essential for airfoil design analysis. The shape of the  $C_P$  curve reveals critical flow phenomena, including the location and strength of shock waves, the onset of flow separation due to adverse pressure gradients, and the extent of laminar flow regions, all of which profoundly impact overall efficiency and operational stability. Consequently, the ability to rapidly and accurately predict the pressure distribution is important for an intelligent agentic workflow. To this end, we employ a neural operator surrogate model to learn the complex mapping from airfoil geometry to its corresponding surface pressure function. Specifically, we utilize a Deep Operator Network (DeepONet), an architecture well-suited for learning operators between infinite-dimensional function

spaces. The DeepONet is trained to map an input function, representing the airfoil geometry via its Class Shape Transformation (CST) parameters and surface coordinates, to an output function, which is the continuous pressure coefficient distribution along the chord,  $C_P(x)$ . Within our multi-agent framework, this predicted  $C_P$  distribution serves as a critical piece of qualitative information for the Systems Engineer agent. This agent assesses a candidate design by comparing its predicted  $C_P$  curve against that of the RAE2822 airfoil, a well-established supercritical benchmark [6]. This qualitative comparison allows the agent to evaluate subjective aerodynamic features, such as the desirability of a 'rooftop' pressure distribution, and subsequently assign a rating to the design. This provides nuanced feedback that guides the design selection process beyond scalar performance metrics. Details related to our DeepONet surrogate model can be found in Appendix A4.3.

## Utility Score

A fundamental challenge in multi-objective engineering design is the comparison of candidates based on several, often conflicting, performance metrics. In airfoil design, for instance, a candidate must be evaluated on its lift ( $C_L$ ), drag ( $C_D$ ), and moment ( $C_M$ ) coefficients, making direct ranking non-trivial. To address this, we employ utility theory, a formal framework for quantifying preferences under uncertainty and multiple objectives [296]. This approach allows us to translate the vector of performance metrics for each design into a single, scalar value known as a utility score. This score represents the overall desirability of the design, enabling direct comparison and ranking. Within our multi-agent framework, this quantitative evaluation is crucial for automation. The engineering agents utilize a utility scoring tool to assess each proposed design and filter out those that do not meet a minimum performance threshold. This threshold is established by the utility score of the RAE2822 airfoil, a standard benchmark, which is calculated to be approximately 0.40 under the defined preference structure. This method provides a clear, consistent, and computationally efficient criterion for automated design selection and iteration.

To compute the overall utility score, we first define individual utility functions for each aerodynamic coefficient, which are then combined through a weighted sum that reflects their relative importance in the design objectives. The combined utility,  $U_{comb}$ , is expressed as:

$$U_{comb} = w_{CL}U(C_L) + w_{CD}U(C_D) + w_{CM}U(C_M) \quad (8.4.1)$$

where the weights are set to  $w_{CL} = 0.5$ ,  $w_{CD} = 0.3$ , and  $w_{CM} = 0.2$ , prioritizing lift, followed by drag and moment. The individual utility functions,  $U(\cdot)$ , are tailored to the specific preference for each metric. For the lift coefficient ( $C_L$ ), where higher values are preferred, we impose a hard constraint for viability ( $C_L \geq 0.5$ ) and model diminishing returns using a concave, square-root function. For the drag coefficient ( $C_D$ ), where lower values are strongly preferred, an exponential decay function heavily rewards low drag and rapidly penalizes any increase. Lastly, for the pitching moment coefficient ( $C_M$ ), considered less critical, a simple linear function maps its typical operational range to a utility score. The specific mathematical formulations for each utility function are detailed below:

$$U(C_L) = \begin{cases} -5.0 & \text{if } C_L < 0.5 \\ \left( \frac{\min[\max(C_L, 0.5), 1.2] - 0.5}{1.2 - 0.5} \right)^{0.5} & \text{if } C_L \geq 0.5 \end{cases} \quad (8.4.2)$$

$$U(C_D) = \exp(-65 \cdot C_D) \quad (8.4.3)$$

$$U(C_M) = \frac{\max[\min(C_M, 0.0), -0.30] - (-0.30)}{0.30} \quad (8.4.4)$$

#### 8.4.6 Filters

In our frameworks, filter serve the purpose of eliminating designs that do not meet a specific user criteria to narrow down the design space for the Design and Systems Engineer agents to operate on. We use two types of filters to eliminate unwanted designs. Below



we explain the two types of filters used in this framework.

### **Utility score filter**

To narrow down the design selection scope to the most promising candidates, our MAS framework employs a utility score filter as a primary mechanism for automated design down-selection. This filter is integrated into the agents' toolbox and is systematically applied each time a new set of designs is generated. The filtering strategy adapts to the design stage, reflecting a progression from broad exploration to focused refinement. In the initial phase, the filter relies on performance coefficients predicted by a low-fidelity, rapid surrogate model such as NeuralFoil. At this stage, the objective is to quickly prune the design space by eliminating any candidates that fail to meet a minimum performance benchmark. This is achieved by discarding all designs with a calculated utility score below a predefined threshold, initially set to 0.4, which corresponds to the performance of the baseline RAE2822 airfoil. As the design process advances into subsequent iterative loops, the evaluation becomes more stringent. The system transitions to using the higher-fidelity Bayesian surrogate model for more accurate coefficient estimation. Concurrently, the filtering criteria are enhanced to incorporate not only the quantitative utility score but also a qualitative assessment of the airfoil's pressure distribution ( $C_P$ ). In these later stages, the agents use the filter to eliminate designs that do not meet both the utility score target and the desired aerodynamic characteristics of the pressure plot when compared against the established benchmark. This dual-criteria approach ensures that surviving designs demonstrate good performance, both in terms of integral coefficients and physically realistic aerodynamic behavior.

### **$\alpha$ -risk filter**

In a set-based design paradigm, where numerous candidate designs are evaluated concurrently, effective filtering mechanisms are essential for focusing effort on the most promising options. While the utility score provides a measure of expected performance, it

does not fully account for the risk stemming from uncertainties, which can be aleatoric (inherent randomness) or epistemic (lack of knowledge), the latter being particularly relevant when using predictions from surrogate models. To ensure the selection of robust designs that are resilient to these uncertainties, a risk-based filtering approach is employed as a secondary check. This method goes beyond mean performance to assess the potential for underperformance. For this purpose, we adopt the Conditional Value at Risk (CVaR) as a coherent risk metric [159]. CVaR measures the expected value of a performance metric given that it falls below a certain quantile of its distribution.

For a given design and its associated lift coefficient ( $C_L$ ) represented by a random variable, the CVaR at a confidence level  $\alpha$  is defined as the conditional expectation of  $C_L$  below the Value at Risk (VaR), which is the  $(1 - \alpha)$ -quantile of its distribution. Mathematically, for a random variable  $X$ ,  $CVaR_\alpha(X) = \mathbb{E}[X|X \geq VaR_\alpha(X)]$ . We estimate CVaR using the historical method, which leverages the probabilistic nature of the Bayesian surrogate by generating a large number of independent samples from its posterior predictive distribution for each design. The CVaR is then calculated as the empirical mean of the worst  $(1 - \alpha)\%$  of these samples. This metric is particularly advantageous for design assessment because, unlike VaR, it captures the magnitude of potential underperformance in the tail of the distribution, not just the probability of a shortfall. This allows the agents to filter out designs that, despite having a favorable expected utility, carry an unacceptably high risk of failing to meet a minimum performance threshold,  $VaR_{CL\_target}$ . Note that since our goal is to identify designs that are greater than a minimum  $C_L$ , we invert the sign of  $VaR_\alpha(CL)$ . The implementation of this filtering step is detailed in Algorithm 2.

---

**Algorithm 2:** CVaR-based Filtering of Candidate Design

---

**Input:** Set of  $N$  designs  $\{d_i\}_{i=1}^N$ , Bayesian surrogate model  $M$ , number of samples

$m$ , confidence level  $\alpha_{CL}$ , performance threshold  $VaR_{CL\_target} = -0.70$

**Output:** A filtered set of robust designs  $D_{robust}$

```
1 begin
2   Initialize:  $D_{robust} \leftarrow \emptyset$ 
3   for  $d_i \in \{d_i\}_{i=1}^N$  do
4     Generate  $m$  i.i.d. samples of lift coefficient  $\{C_{L,i}^{(j)}\}_{j=1}^m$  using the posterior
       predictive distribution from surrogate  $M$  for design  $d_i$ 
5     Sort the samples in ascending order:  $C_{L,i}^{(1)'} \leq C_{L,i}^{(2)'} \leq \dots \leq C_{L,i}^{(m)'}$ 
6     Determine the number of tail samples,  $k \leftarrow \lfloor (1 - \alpha_{CL}) \cdot m \rfloor$ 
7     Calculate the CVaR for the design:  $CVaR_i \leftarrow \frac{1}{k} \sum_{j=1}^k C_{L,i}^{(j)'}$ 
8     if  $CVaR_i \leq VaR_{CL\_target}$  then
9       Add  $d_i$  to  $D_{robust}$ 
10    end
11  end
12  return  $D_{robust}$ 
13 end
```

---

## 8.5 Workflow and Experiments

In this section, we describe the different stages of our workflow, referenced in figure 8.1 and results from our experiments.

### 8.5.1 Stage 0: Tool development

The initial phase of our methodology, the Precursor Stage, is dedicated to the development and validation of a custom suite of software tools that form the functional core of the multi-agent system. This foundational stage follows a synergistic human-in-the-loop

model, the ‘Manager’, collaborates with the Coding Assistant. The Manager architects the entire engineering workflow, defining the precise specifications for all necessary tools, while the Coding Assistant translates these high-level instructions into executable code. The development process is inherently iterative, involving a continuous feedback loop where the Manager verifies the generated tools, often requiring two to three cycles of refinement to achieve the final, validated code. This dynamic was particularly evident in the creation of complex components, such as the neural network surrogates. While the Coding Assistant successfully generated an initial codebase, achieving optimal predictive accuracy necessitated direct intervention from the Manager, who leveraged their expertise to modify key hyperparameters and make subtle architectural adjustments. This experience underscores a key aspect of our approach: AI assistants can significantly accelerate the implementation of well-defined software, while the nuanced judgment and deep expertise of human specialists remain indispensable for complex, performance-sensitive tasks.

### 8.5.2 Stage 1: User request

The first stage of the workflow involves the Design agent receiving a design request from the human manager. In this study, the request to the agent is defined by a need to design airfoils with maximum lift coefficient possible at the given flow conditions. Note here that the framework supports other flow conditions within the range described in table A9. Also, changing design goals is possible, with modifications required in the tool set to ensure appropriate utility functions and filtering process.

#### User request for initializing workflow

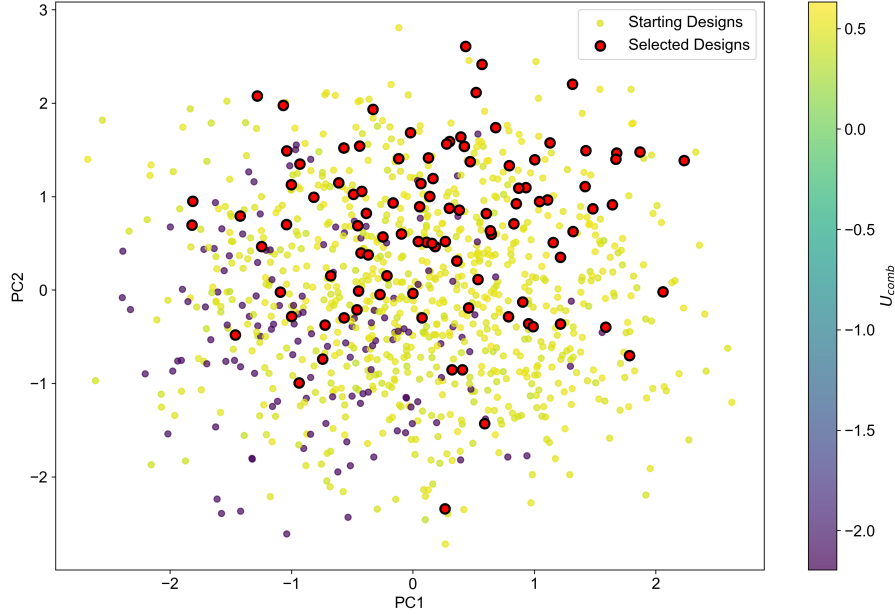
"Create airfoil designs for the following flow conditions: Mach=0.6 at an angle of attack of 2.5 degrees. Assume a Reynold's number of 6.3 million for this problem. Your goal should be to maximize the lift coefficient of the airfoil taking into consideration other aspects such as coefficient of drag, moment coefficient, and pressure distribution over the airfoil surface"

### 8.5.3 Stage 2: Design generation phase

The second stage of the workflow initiates the automated design generation and evaluation cycle, orchestrated by the Design agent. The primary objective of this phase is to efficiently explore the vast design space and identify an initial set of high-potential airfoil candidates. The process commences with the design agent systematically sampling a large set of design vectors from the predefined parameter space. Each sampled parameter set is then processed by the ‘AirFoil Generator’ tool, which translates the abstract CST coefficients into the Cartesian coordinates defining a unique airfoil geometry. NeuralFoil surrogate is then used for rapidly predicting estimates for the key aerodynamic performance metrics: the coefficients of lift ( $C_L$ ), drag ( $C_D$ ), and moment ( $C_M$ ), for each generated airfoil. Subsequently, the vector of predicted coefficients for each design is fed into the ‘Utility Score’ tool, which converts the multi-objective performance data into a single, scalar utility score ( $U_{comb}$ ) as previously detailed in equation 8.4.4. This score provides a quantitative measure of the design’s overall desirability. The final step in this stage is a critical down-selection process: the system applies a filter that automatically discards any design with a utility score below a predefined threshold of 0.4. This threshold is strategically chosen as it corresponds to the utility score of the benchmark RAE2822 airfoil, ensuring that only designs predicted to outperform the baseline are retained. In addition to this filter, we further down-sample this set by selecting the top 100 designs with highest  $U_{comb}$  score. This automated filtering mechanism drastically reduces the initial candidate pool to a smaller, more manageable subset of promising designs, which then proceed to the subsequent stages of the workflow for further modification. Figure 8.3 shows the resulting subset of designs after the utility score-based filter has been applied.

### 8.5.4 Stage 3: Sensitivity analysis

Following the initial design generation and filtering, the workflow transitions to a crucial analytical phase orchestrated by the Analyst agent. The primary objective of



**Figure 8.3:** Plot showing the starting design set and the selected designs (in red) after the utility score-based filtering process. X and Y axes show the Principal Components of the CST parameters that define airfoil geometry. Principal Component values are used here for visualization purpose.

this stage is to move beyond merely identifying performant designs to fundamentally understanding the relationship between the design variables and the resulting aerodynamic performance. Given that the mapping from the nine-dimensional CST parameter space  $\in \mathbb{R}^9$  to the aerodynamic coefficients  $(C_L, C_D, C_M)$  is highly non-linear and non-intuitive, a systematic analysis is essential to enable intelligent design refinement. To achieve this, the Analyst agent is tasked with performing a global sensitivity analysis (GSA). For this purpose, a variance-based method utilizing Sobol indices was selected, as it excels at quantifying the influence of input parameters in complex models by accounting for both their individual effects and their interactions, a significant advantage over local, one-at-a-time sensitivity measures [158]. The ultimate goal is to distill the quantitative results of the GSA into a set of qualitative, actionable guidelines that can be used by the Design Agent to strategically modify airfoil geometries in subsequent stages.

To provide a quantitative foundation for these guidelines, the Sobol method decomposes the total variance of a model output,  $V(Y)$ , into fractions attributable to each input parameter and their interactions. For a model  $Y = f(X_1, \dots, X_D)$ , the first-order Sobol

index ( $S_i$ ) for an input  $X_i$  is defined as

$$S_i = V[E(Y|X_i)]/V(Y), \quad (8.5.1)$$

which represents the main effect of  $X_i$  on the output variance. Higher-order interactions are also captured; for instance, the second-order index,

$$S_{ij} = (V[E(Y|X_i, X_j)] - V[E(Y|X_i)] - V[E(Y|X_j)]) / V(Y) \quad (8.5.2)$$

quantifies the portion of variance due to the interaction between  $X_i$  and  $X_j$  alone. To capture the full influence of a parameter, including all its interactions, the total-effect index ( $S_{Ti}$ ) is computed as

$$S_{Ti} = E[V(Y|X_{\sim i})] / V(Y) \quad (8.5.3)$$

where  $X_{\sim i}$  denotes the set of all input parameters except  $X_i$  [158]. In our implementation, the Analyst agent applies this method to the  $P = 9$  CST parameters for each of the three performance metrics ( $C_L, C_D, C_M$ ). These metrics were obtained using NeuralFoil surrogate. A set of  $N = 128$  base samples is drawn from the design space, and following the Saltelli sampling scheme, a total of  $N(2P + 2) = 2,560$  analysis points are evaluated. The agent then computes the first-order and total-effect indices, generating both graphical plots (figures A13, A12) and quantitative reports. Finally, the agent processes this information to synthesize a set of strategic recommendations, such as the report shown in figure 8.4. These quantitative guidelines provide a structured, data-driven basis for design modification, moving the agent from a purely heuristic to an analytical approach and thereby enabling more efficient and targeted design improvement.

### Sensitivity analysis report from Analyst

- **CL analysis:** Based on the Sobol analysis, the following CST parameters are most significant for the Coefficient of Lift (CL)
  - CST\_U2: This is a highly sensitive parameter. The influence plot shows a positive correlation, meaning \*increasing CST\_U2 will increase CL\*.
  - CST\_L3: This is another highly sensitive parameter. The influence plot shows a negative correlation, meaning \*increasing CST\_L3 will decrease CL\*.
  - CST\_L2: This parameter has high sensitivity. The influence plot indicates a negative correlation, so \*increasing CST\_L2 will decrease CL\*.
  - CST\_U3: This parameter has a significant impact. The influence plot shows a positive correlation, meaning \*increasing CST\_U3 will increase CL\*.
- **CD analysis:** Based on the Sobol analysis, the following CST parameters are most significant for the Coefficient of Drag (CD)
  - CST\_L3: This is the most dominant parameter affecting CD. The influence plot shows a strong positive correlation, meaning \*increasing CST\_L3 will significantly increase CD\*.
  - CST\_U3: This parameter is highly sensitive. The influence plot shows a negative correlation, so \*increasing CST\_U3 will decrease CD\*.
  - CST\_U4: This parameter is also highly sensitive. The influence plot indicates a negative correlation, meaning \*increasing CST\_U4 will decrease CD\*.
  - CST\_U2: This parameter has a moderate but notable sensitivity. The influence plot shows a negative correlation, meaning \*increasing CST\_U2 will decrease CD\*.
- **CM analysis:** Based on the Sobol analysis, the following CST parameters are most significant for the Coefficient of Moment (CM)
  - CST\_L3: This is the most sensitive parameter for CM. The influence plot shows a positive correlation, meaning \*increasing CST\_L3 will increase CM\* (make it less negative).
  - CST\_U2: This parameter is highly sensitive. The influence plot indicates a positive correlation, so \*increasing CST\_U2 will increase CM\*.
  - CST\_U3: This parameter has a high sensitivity. The influence plot shows a negative correlation, meaning \*increasing CST\_U3 will decrease CM\*.
  - CST\_L2: This parameter is also highly sensitive. The influence plot shows a positive correlation, meaning \*increasing CST\_L2 will increase CM\*.

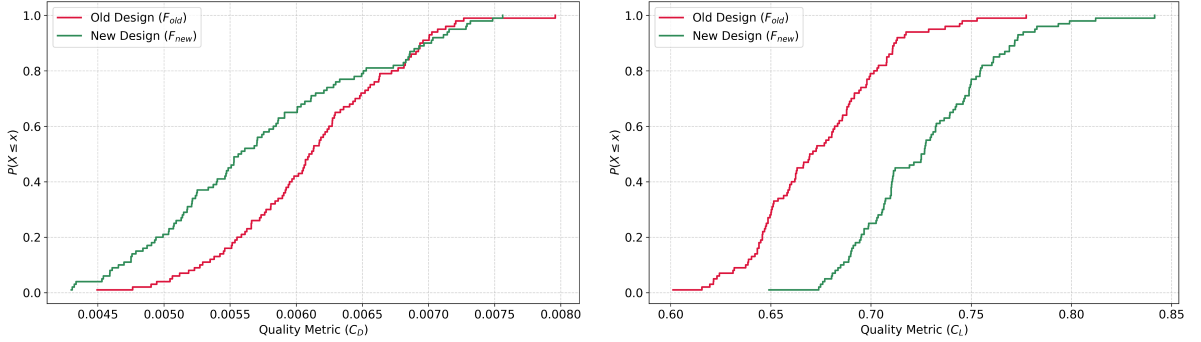
**Figure 8.4:** Sensitivity analysis report generated by the Analyst agent using results from Sobol analysis. This heuristic is used by the Design agent for design modifications  $\mathcal{M}_k$ .

### 8.5.5 Stage 4: Design revision and risk-based filtering

Upon completion of the sensitivity analysis, the workflow advances to a directed design refinement stage, where the Design agent’s objective is to enhance the performance of the 100 promising candidates selected in Stage 2. Leveraging the quantitative guidelines from the Analyst agent’s sensitivity report, the Design agent systematically modifies the CST parameters of each airfoil. These modifications are targeted at the parameters identified as having the most significant influence on the aerodynamic coefficients, with the explicit goals of increasing the lift coefficient ( $C_L$ ), reducing the drag coefficient ( $C_D$ ), and improving the pitching moment ( $C_M$ ). As illustrated in the comparative Cumulative Distribution Function (CDF) plots shown in the figure 8.5, this guided refinement strategy successfully shifts the population of designs towards improved overall performance, particularly in



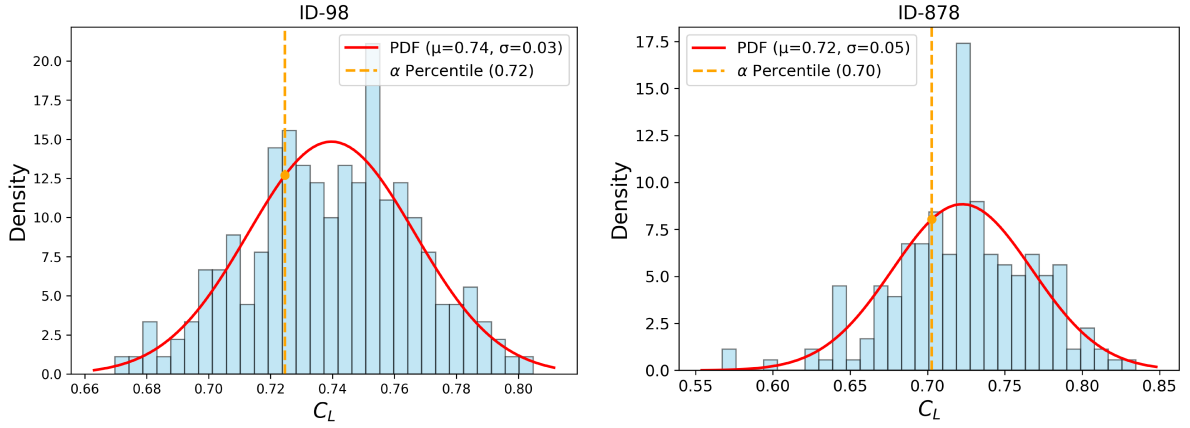
$C_L$  and  $C_D$ . However, we also observe a common challenge, where improvements in lift and drag can sometimes have an adverse effect on the pitching moment. Following these modifications, the ‘Airfoil Generator’ tool is used to create the updated airfoil geometries. At this stage, we transition from the ‘NeuralFoil’ model to a custom-trained Bayesian surrogate for performance evaluation. This change is motivated by two key advantages: the Bayesian surrogate offers a more reliable estimate of  $C_D$  within our specific design space, and more importantly, it provides a full posterior predictive distribution for each aerodynamic coefficient.



**Figure 8.5:** Comparison between cumulative distribution of  $C_D$  and  $C_L$  before and after design modification by the Design agent. The cumulative distribution shows the improvement in overall values of  $C_D$  and  $C_L$  after parameter modification based on the Sensitivity analysis.

The generation of refined designs is followed by a subsequent filtering step to ensure that the selected candidates are not only high-performing but also robust to the inherent uncertainties of the surrogate models. With the probabilistic performance estimates provided by the Bayesian surrogate, we now implement our risk-based filtering strategy. This approach moves beyond considering only the mean predicted performance and explicitly addresses the risk due to epistemic uncertainty in the model. Specifically, the newly generated designs are subjected to the  $\alpha$ -risk filter, which implements the Conditional Value-at-Risk (CVaR) methodology detailed in Algorithm 2. For this filtering process, we define a confidence level of  $\alpha = 0.7$  and a target Value at Risk for the lift coefficient of  $Var_{CL} = -0.70$ . This configuration ensures that for a design to be selected, its expected  $C_L$  in the worst 30% of predicted outcomes must be greater than or equal to 0.70. Figure 8.6 shows the  $C_L$  distribution of two design candidates; ID-98 which was

selected as valid during the risk-filtering process whereas ID-878 was removed, based on CVaR value. By imposing this stringent condition, the system effectively mitigates the risk of selecting designs that appear promising due to favorable but uncertain surrogate predictions. This process prunes the set of modified designs, retaining only those that demonstrate robust performance, thereby increasing confidence in the final set of candidates passed to subsequent stages. A total of 64 designs remain at the end of Stage 4.



**Figure 8.6:** Plot showing  $C_L$  distributions of two design candidates: ID-98 (retained) and ID-878 (removed) used for  $\alpha$ -risk filtering strategy. The orange line marks the  $C_L$  value at  $1 - \alpha$  percentile of the distribution.

### 8.5.6 Stage 5: Design Revision and evaluation of Pressure distribution

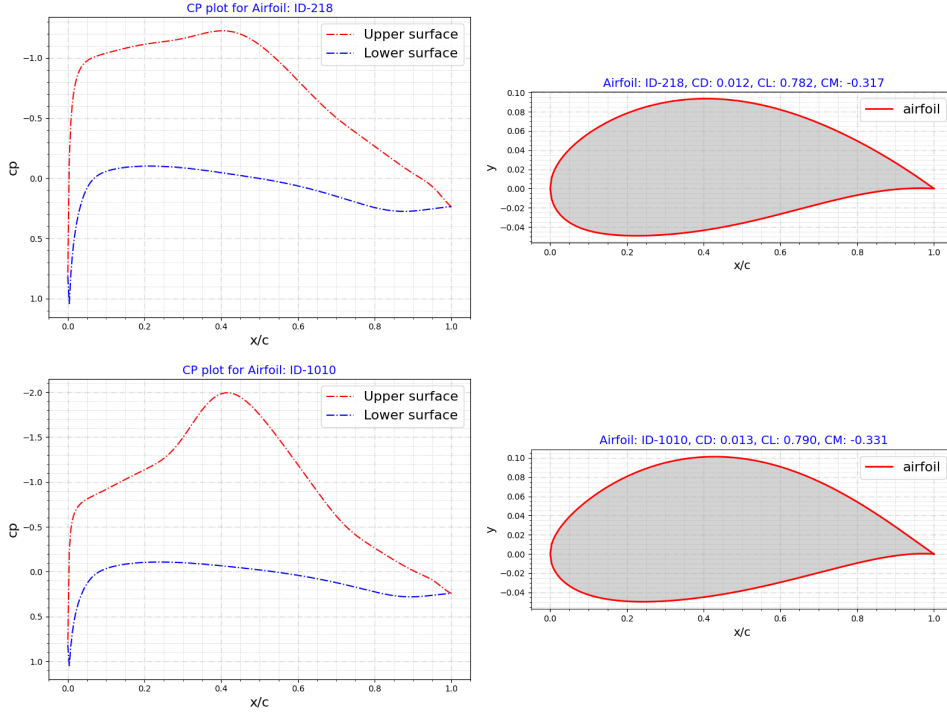
In stage 5, the Design agent operates on the subset of designs identified as robust to epistemic uncertainty at the conclusion of Stage 4. This phase represents a shift towards fine-tuning an already promising and reliable set of candidates. The Design agent initiates a second round of modifications, once again leveraging the quantitative insights from the sensitivity analysis report generated in Stage 3. However, unlike the broader refinement in the previous stage, these modifications are more focused, with the primary objective of further enhancing the lift coefficient ( $C_L$ ) while maintaining the favorable drag and moment characteristics of the robust designs. Following each modification, the ‘Airfoil Generator’ tool produces the new geometry, which is then evaluated using the Bayesian

surrogate. For this performance assessment, the point estimate for each aerodynamic coefficient is taken as the mean of the posterior predictive distribution, calculated across 200 independent and identically distributed (iid) ensembles. This is followed by a final round of utility scoring and filtering, mirroring the process from Stage 2. The refined designs are ranked in descending order based on their new utility scores, and the top 50 designs are retained, forming the subset for detailed review.

With the final set of 50 robust designs established, the analysis transitions from evaluating integral aerodynamic coefficients to a more detailed characterization of the flow physics. This is crucial because integral values like  $C_L$  and  $C_D$  do not fully capture the nuanced aerodynamic behavior of an airfoil. To facilitate this, the framework employs a specialized Deep Operator Network (DeepONet) surrogate, discussed earlier in section 8.4.5. The Design agent utilizes this tool to generate a  $C_P$  plot for each of the 50 selected designs. A sample of this plot generated for all 50 design candidates is shown in figure 8.7. The analysis of these distributions is important for a comprehensive design assessment, as it allows for the qualitative evaluation of critical flow features. This includes identifying the location and strength of shock waves, assessing the adversity of the pressure gradient on the aft portion of the airfoil which can indicate a propensity for flow separation, and ensuring a smooth pressure recovery. The generation of these detailed  $C_P$  plots provides the necessary physical insights, which, along with the final performance metrics, serve as the comprehensive input for the subsequent automated design review and final selection stage.

### 8.5.7 Stage 6: Automated design review and filtering

Following the generation and detailed characterization of the top 50 designs in the preceding stage, the workflow transitions to a rigorous vetting process. This phase is executed by a specialized ‘Systems Engineer’ agent, whose primary function is to perform a holistic evaluation that emulates the judgment of an experienced human



**Figure 8.7:** Coefficient of Pressure plots for two airfoil samples generated after analysis using the DeepONet surrogate. These pressure plots allows the Systems Engineer agent to perform a comprehensive review of the airfoil design, to include choice of design parameters, integral coefficients such as  $C_D$ ,  $C_L$ , and  $C_M$ , and the pressure distribution curves generated here.

engineer. The agent’s task is to holistically assess the designs based on a combination of quantitative metrics and qualitative aerodynamic features. This is achieved through a two-pronged evaluation strategy: first, by re-evaluating the utility scores for the aerodynamic coefficients ( $C_D$ ,  $C_L$ ,  $C_M$ ), and second, by assigning a qualitative rating to the airfoil’s pressure distribution ( $C_P$ ) curve. The overarching goal of this stage is to automate the final down-selection by systematically applying a set of heuristic rules to classify each of the 50 candidates as either ‘valid’ or ‘invalid’, thereby filtering out designs that are numerically optimal but aerodynamically or structurally impractical.

The evaluation rubric for the Systems Engineer agent is structured around a direct comparison with the RAE2822 benchmark airfoil (shown in Appendix figure A14), whose performance at the specified flow conditions is characterized by  $C_D = 0.010$ ,  $C_L = 0.522$ , and  $C_M = -0.073$ , corresponding to utility scores of  $U(C_D) = 0.518$ ,  $U(C_L) = 0.177$ ,  $U(C_M) = 0.7566$ , and a combined score of  $U_{comb} = 0.3955$ . The qualitative assessment of the  $C_P$  curve is performed using an ordinal rating on a scale of 1 (worst) to 5 (best),

with the benchmark’s pressure distribution assigned a baseline rating of 3. The agent is instructed to penalize designs exhibiting aerodynamically poor features; for instance, a significant peak or bump on the upper surface pressure distribution between 30% and 60% of the chord: a typical indicator of a strong, undesirable shock wave results in a low rating of 1 or 2. This automated review also includes a check against non-functional requirements such as manufacturability and structural integrity by comparing the candidate’s geometry against the standard shape of the RAE2822. This multi-faceted analysis culminates in a single binary classification, governed by a strict rule: a design is deemed ‘Valid’ only if its combined utility score is greater than or equal to 0.41 and its pressure distribution rating is 3 or higher. The specific instructions provided to the agent are summarized in figure 8.8.

**Design evaluation rules for Systems Engineer agent**

1. **Pressure Distribution ( $C_P$ ) Rating (Scale 1-5):** Compare the sample’s  $C_P$  plot to the benchmark. The benchmark is rated 3. A rating of 3 or higher is assigned if the sample shows no significant adverse departure. A pronounced peak on the upper surface near the mid-chord ( $0.3 < x/c < 0.6$ ) must be rated 1 or 2 based on severity.
2. **Non-Functional Requirements:** Assess the airfoil shape for practical viability (e.g., manufacturability, structural integrity) using the benchmark shape as a reference.
3. **Performance Coefficient Comparison:** Quantitatively compare the sample’s  $C_L$ ,  $C_D$ , and  $C_M$  values against the benchmark ( $C_D = 0.01$ ,  $C_L = 0.522$ ,  $C_M = -0.073$ ).
4. **Utility Score Comparison:** Compare the sample’s combined utility score ( $U_{comb}$ ) against the benchmark’s score of 0.3955.
5. **Final Validity Decision:** A design is classified as **Valid** if and only if:
  - $U_{comb} \geq 0.41$
  - $C_P \text{ Rating} \geq 3$

**Figure 8.8:** *Instruction set provided to the Systems Engineer during the automated design evaluation and filtering phase.*

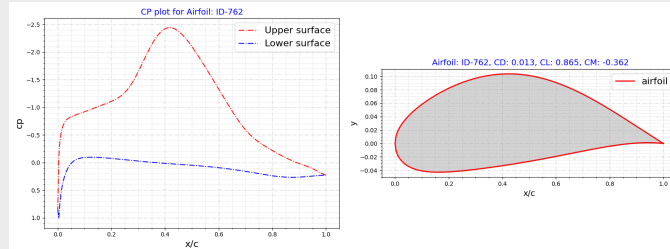
In practice, the Systems Engineer agent systematically processes each of the 50 final design candidates, applying the defined rubric to generate the utility scores and ordinal  $C_P$  ratings. This automated procedure ensures a consistent and objective application of the evaluation criteria across the entire design cohort. Subsequently, the agent applies the final filtering rule, partitioning the set of 50 designs into two distinct groups: a smaller subset of ‘valid’ designs that satisfy both the performance and aerodynamic quality thresholds, and a group of ‘invalid’ designs that are discarded in next steps. This automated rating

and filtering step is a critical component of the multi-agent framework, as it codifies expert-level judgment into a repeatable process, thereby saving significant man-hours that would otherwise be spent on manual review. It is important to note that the agent's role in this final stage is purely evaluative. It does not generate feedback for design improvement; rather, its purpose is to deliver a refined, high-quality subset of designs for final consideration by the human design team. Figures 8.9 and 8.10 show examples of responses generated by the Systems Engineer agent during this automated review process, to filter out designs.

#### Design evaluation for Design ID-762 (Invalid)

"The pressure distribution for ID-762 shows a very strong suction peak on the upper surface around  $x/c = 0.4$ , reaching approximately  $-2.5$  Cp. This peak is significantly more pronounced and located further aft compared to the benchmark RAE2822. The lower surface pressure is relatively flat and slightly negative over a large portion, which differs from the more positive pressure observed on the benchmark's lower surface. This strong mid-chord suction peak is a concern for potential shock formation and increased drag at higher speeds. The airfoil ID-762 is noticeably thicker and more highly cambered than the RAE2822 benchmark. This increased thickness generally contributes to better structural integrity. The smooth contour suggests good manufacturability and maintainability, although the higher camber might require specific manufacturing considerations. The airfoil ID-762 achieves a significantly higher coefficient of lift ( $CL = 0.865$ ) compared to the benchmark ( $CL = 0.522$ ). However, it also exhibits higher drag ( $CD = 0.013$  vs  $0.010$ ) and a substantially larger nose-down pitching moment ( $CM = -0.362$  vs  $-0.073$ ), which could pose stability challenges. The combined utility score for ID-762 is  $0.4898$ , which is higher than the benchmark's  $0.3955$ . This is primarily driven by a very high utility for lift ( $util(CL) = 0.7217$ ). However, the utility for pitching moment ( $util(CM) = 0.0$ ) is very low, indicating a significant penalty for the large negative moment, and the utility for drag ( $util(CD) = 0.4297$ ) is also lower than the benchmark."

Pressure rating: 2; Valid: False



**Figure 8.9:** Example of feedback generated by the Systems Engineer during automated design evaluation stage for an invalid design.

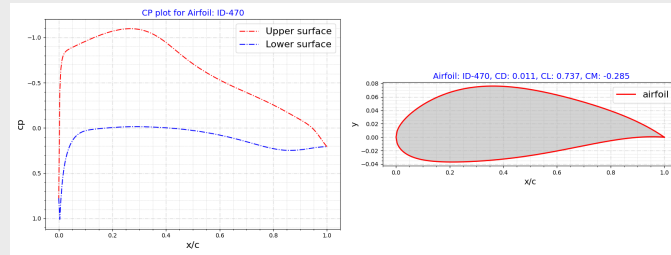
## 8.5.8 Iterative Human-in-the-Loop Design Review and Refinement

After the automated design review and filtering stage, the workflow enters a final, collaborative phase of iterative design revision and review. This stage is initiated with the small subset of designs classified as 'Valid' by the Systems Engineer agent in the

### Design evaluation for Design ID-470 (Valid)

“The pressure distribution for airfoil ID-470 exhibits a smooth and well-behaved curve on both upper and lower surfaces, with no significant adverse pressure gradients or sudden changes, indicating good flow characteristics. The suction peak on the upper surface is less pronounced than the benchmark RAE2822, which contributes to its higher lift. The airfoil profile is robust with smooth contours and a relatively thick section, suggesting good manufacturability, structural integrity, and no obvious maintenance issues. Aerodynamically, ID-470 generates significantly more lift ( $CL = 0.737$ ) than the benchmark ( $CL = 0.522$ ), leading to a much higher lift utility. However, it also has a slightly higher drag ( $CD = 0.011$  vs  $0.010$ ) and a considerably larger nose-down pitching moment ( $CM = -0.285$  vs  $-0.073$ ), which severely reduces its moment utility. Despite the low moment utility, the high lift performance results in a combined utility score ( $0.452$ ) that surpasses the benchmark ( $0.396$ ).”

Pressure rating: 4; Valid: True



**Figure 8.10:** Example of feedback generated by the Systems Engineer during automated design evaluation stage for a valid design.

preceding step. The core objective shifts from broad exploration and automated filtering to a highly focused, human-supervised refinement process. This human-in-the-loop approach is critical for preventing 'runaway' design choices, where an autonomous system might over-optimize for a specific metric at the expense of unquantified but crucial engineering characteristics. The process involves a tight feedback loop between the Design agent, the Systems Engineer agent, and a human Manager, ensuring that the final designs are not only performant according to the defined user goals but also align with the nuanced, often tacit, requirements of a practical engineering application.

Each iteration within this phase begins with the Design agent modifying a candidate airfoil. The agent's modifications are informed by a combination of two key inputs: the global sensitivity analysis report, which provides a foundational understanding of the parameter-performance relationship, and specific, targeted feedback generated from the previous review cycle. Once a design is modified, its new geometry and predicted pressure distribution are presented for a joint review by the Systems Engineer agent and the human Manager. The Systems Engineer agent first performs an assessment analogous to its role in Stage 6, recalculating the combined utility score and assigning a qualitative

rating to the pressure curve. This provides a consistent, data-driven baseline for the review. However, the final authority in this stage rests with the human Manager, who can override the agent's assessment. If the Manager approves a design as a candidate for further improvement, the Systems Engineer agent is tasked with generating specific, actionable feedback. This feedback leverages both the sensitivity data and the agent's analysis of the current design's shortcomings to guide the Design agent's next modification (e.g., "Reduce the adverse pressure gradient post-shock by adjusting the aft-camber parameters"). Figures 8.11 – 8.12 show the feedback generated by the Systems Engineer for the first two design loops for a particular design that remains 'Valid' till the end of the design cycle. Additional responses for the third and fourth iteration can be found in figures A15 and A16 in Appendix. Note that after every feedback, the Design agent modifies the design to improve the particular aspect of design where design improvements were suggested. In general, all the designs selected till the end exhibit high  $C_L$  values but suffer from negative  $C_M$  values, hence the design improvement strategy mostly focuses on improving  $C_M$  in the iterative stage.

This iterative cycle of modification, joint review, and feedback generation continues until a final, small subset of approximately four–five designs is identified, meeting the satisfaction of the human Manager. The successive filtering of designs is shown in figure 8.13. The determination of design validity in this phase is explicitly deferred to human judgment, given the significantly reduced number of candidates, which makes manual review non-labor-intensive. Once this set of designs is finalized, the Manager has the option to trigger a final verification step to confirm the suitability of the candidates with a numerical CFD model. This serves as a crucial final validation, ensuring that the performance predicted by the surrogate models translates accurately into a physics-based simulation environment before final design selection.

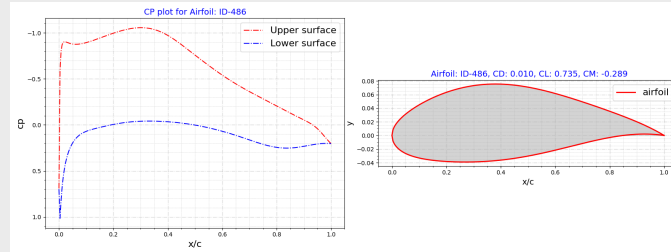


## Design feedback for Design ID-486 - iteration 1

“The pressure distribution for airfoil ID-486 shows a generally similar trend to the RAE2822 benchmark, with an initial acceleration to low pressure on the upper surface. However, there is a slight irregularity or less smooth pressure recovery on the upper surface between  $x/c = 0.2$  and  $x/c = 0.4$ , which is a minor departure from the benchmark’s smoother curve. The lower surface pressure distribution is comparable to the benchmark, with a slightly more pronounced pressure minimum near the leading edge. Overall, the pressure distribution is acceptable. The airfoil shape of ID-486 appears robust and well-proportioned, similar in general characteristics to the RAE2822 benchmark. Visually, it does not present any features that would raise concerns regarding manufacturability, structural integrity, or maintenance. The thickness distribution and camber seem appropriate for an efficient airfoil design.

Comparing aerodynamic performance, the coefficient of drag (CD) for ID-486 is slightly higher than the benchmark (0.01038 vs 0.010), resulting in a slightly lower utility score for drag (0.509 vs 0.518). The coefficient of lift (CL) is significantly higher for ID-486 (0.735 vs 0.522), leading to a much better utility score for lift (0.579 vs 0.177). The coefficient of moment (CM) for ID-486 is considerably more negative (-0.289 vs -0.073), which severely impacts its utility score for moment (0.0375 vs 0.7566), indicating a strong nose-down pitching moment. Despite the poor CM utility, the high CL utility helps the combined utility score of ID-486 (0.450) to be higher than the benchmark (0.3955).”

Pressure rating: 3; Valid: True



**Figure 8.11:** Feedback for design ID-486 generated by the Systems Engineer with human Manager during first iteration.

### 8.5.9 Stage 7: CFD simulation with OpenFoam

The iterative design cycle culminates when the Manager identifies a small, curated subset of designs deemed suitable for the application. At this juncture, the workflow transitions from rapid, surrogate-based evaluation to high-fidelity physical verification. The Manager formally requests a Computational Fluid Dynamics (CFD) evaluation for each of the final candidate airfoils. In response, the Systems Engineer agent initiates a series of automated simulation tasks using the open-source OpenFOAM framework. The core of this analysis is the ‘simpleFoam’ solver, a steady-state solver for incompressible and compressible flows. The simulations solve the Reynolds-Averaged Navier-Stokes (RANS) equations to model the fluid dynamics. Specifically, for a compressible fluid treated as a perfect gas, the solver iteratively finds a solution to the steady-state continuity and

## Design feedback for Design ID-486 - iteration 2

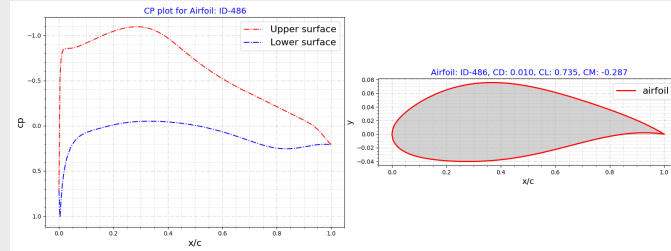
**“Pressure Distribution:** The airfoil ID-486 exhibits a strong suction peak on the upper surface near the leading edge, reaching  $C_p$  values below -1.0, which contributes to high lift. However, there is a noticeable ‘bump’ around  $x/c = 0.2$ , indicating potential local flow acceleration or a weak shock, and the pressure recovery on the upper surface is less smooth compared to the benchmark RAE2822. The lower surface pressure distribution is generally smooth and positive. Airfoil Profile: Airfoil ID-486 appears thicker and has more pronounced camber than the benchmark RAE2822, particularly in the mid-chord region. The leading edge radius seems larger. The overall shape appears structurally sound and manufacturable, similar to the benchmark, without any extreme features that would raise immediate concerns regarding manufacturability, structural integrity, or maintenance.

**Aerodynamic Performance & Utility:** The airfoil ID-486 achieves a significantly higher Coefficient of Lift ( $CL = 0.735$ ) compared to the benchmark ( $CL = 0.522$ ), resulting in a much higher CL utility score (0.5800 vs 0.177). The Coefficient of Drag ( $CD = 0.010$ ) is comparable to the benchmark ( $CD = 0.010$ ), leading to a similar CD utility score (0.5104 vs 0.518). However, the Coefficient of Moment ( $CM = -0.287$ ) is substantially more negative than the benchmark ( $CM = -0.073$ ), resulting in a very low CM utility score (0.0425 vs 0.7566). Despite the poor CM utility, the high CL utility leads to a higher combined utility score (0.4516) for ID-486 compared to the benchmark (0.3955).

### Design Improvements:

1. Increase CST\_U2: This is the most impactful change. Increasing ‘CST\_U2’ will simultaneously increase CM (making it less negative), increase CL, and decrease CD, thereby improving the overall combined utility and pitching moment characteristics.
2. Carefully decrease CST\_U3: To further improve CM (make it less negative), a controlled decrease in ‘CST\_U3’ can be explored. However, this must be balanced against its negative impact on CL and positive impact on CD.
3. Consider increasing CST\_L2: A moderate increase in ‘CST\_L2’ could help increase CM. This would come at the cost of decreasing CL, so it should be considered if the current high CL can be slightly reduced for a more favorable pitching moment.

Pressure rating: 3; Valid: True ”



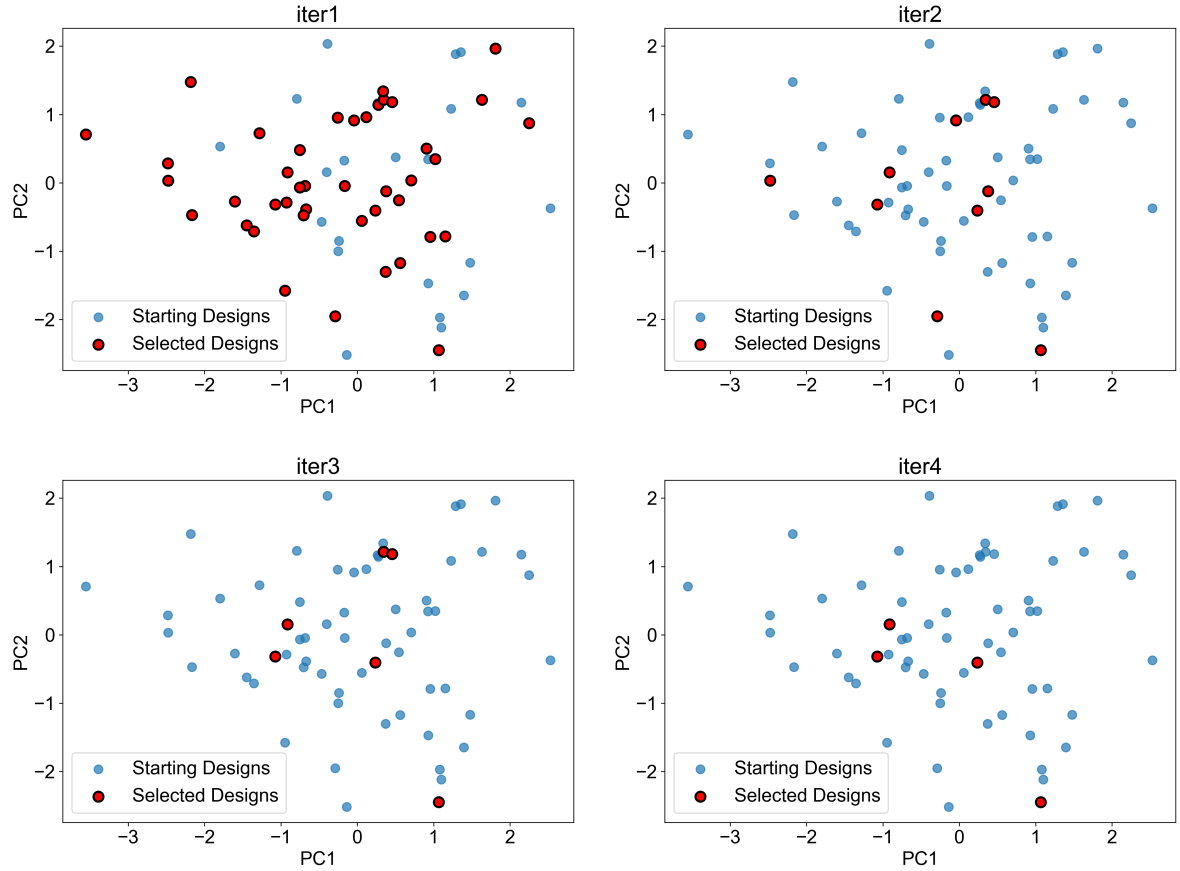
**Figure 8.12:** Feedback for design ID-486 generated by the Systems Engineer with human Manager during second iteration after design modification.

momentum equations:

$$\nabla \cdot (\rho \mathbf{U}) = 0 \quad (8.5.4)$$

$$\nabla \cdot (\rho \mathbf{U} \mathbf{U}) = -\nabla p + \nabla \cdot \boldsymbol{\tau}_{eff} \quad (8.5.5)$$

Here,  $\rho$  is the fluid density,  $\mathbf{U}$  is the mean velocity vector,  $p$  is the mean pressure, and  $\boldsymbol{\tau}_{eff}$  is the effective stress tensor, which includes both molecular and turbulent stresses. To close this system of equations, turbulence is modeled using the one-equation Spalart-Allmaras model, a choice well-suited for external aerodynamic applications involving



**Figure 8.13:** Plot showing the design filtering during iterative design and review process. The  $x$  and  $y$ -axes show the two principal components for the CST parameters for each design. In the iterative phase, the designs are filtered based on the human Manager’s input along with feedback received from the Systems Engineer agent.

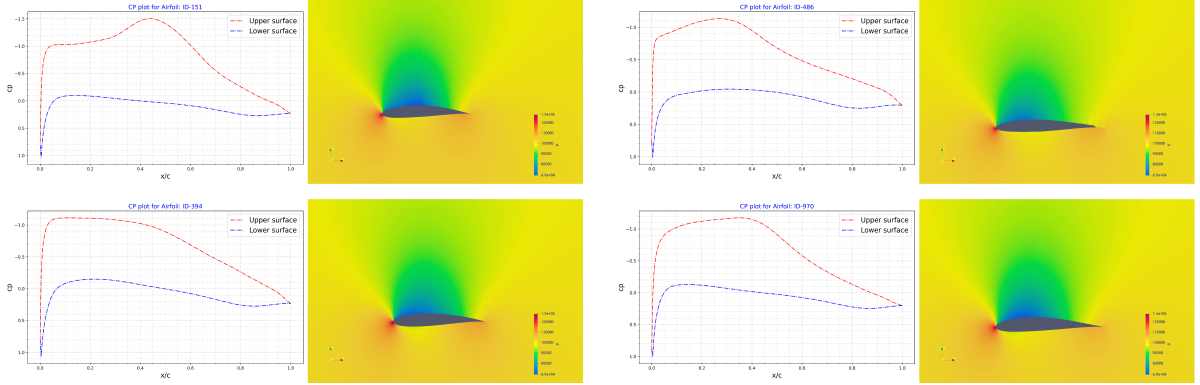
attached boundary layers, as is common in airfoil analysis. This model introduces a transport equation for a modified turbulent kinematic viscosity, thereby providing the turbulent viscosity term required to compute the turbulent stresses within  $\tau_{eff}$ .

To facilitate an automated and repeatable simulation process for various airfoil geometries, a series of scripts developed in prior work were used to handle the mesh generation pipeline. The process begins with a Python script that converts the 2D coordinates of a given airfoil design into a 3D surface geometry file in the Wavefront OBJ format, providing a small extrusion in the spanwise direction. This OBJ file serves as the geometric definition of the airfoil surface for the meshing utility. The subsequent meshing is handled by OpenFOAM’s native ‘blockMesh’ tool, which is configured via a templated ‘blockMeshDict’ file. This dictionary defines a structured C-grid topology around the airfoil, a standard and

efficient approach for resolving the key flow features in external aerodynamics. The mesh is parameterized to allow for programmatic updates; another Python script automatically modifies the ‘blockMeshDict’ to adjust vertex locations corresponding to the leading edge, trailing edge, and points of maximum thickness for each new airfoil design. Significant mesh grading is applied, particularly in the direction normal to the airfoil surface, to achieve a high density of cells within the boundary layer, which is critical for accurately predicting skin friction drag and flow separation characteristics.

Upon completion of the CFD simulations for all selected candidates, the high-fidelity results are presented to the Manager for final review and down-selection. Figure 8.14 shows a comparison between coefficient of pressure distribution predicted by the agentic framework and the corresponding CFD pressure distribution for the final four design candidates. Additional velocity field plots for these candidates can be found in Appendix figure A17, which are used for qualitative analysis by the Manager. This final assessment focuses on comparing the pressure distributions and integral aerodynamic coefficients ( $C_L$ ,  $C_D$ ,  $C_M$ ) against the predictions from the surrogate models used in the earlier stages. This step serves two critical functions: first, it provides validation of a design’s performance in a physics-based environment, and second, it verifies the accuracy of the surrogate models used throughout the design process. For instance, in one review, design ID-151 was discarded due to a strong suction peak observed near the mid-chord in its CFD-computed pressure plot, a feature that was hinted at by the surrogate’s  $C_P$  prediction but was confirmed to be unacceptable by CFD analysis. Similarly, ID-486 was deemed as invalid due to its low  $C_L$  value, although this design has the best  $C_M$  value amongst this lot. The remaining two design candidates exhibited favorable aerodynamic performance and pressure distributions, aligning well with the surrogate predictions, and were thus retained for further development. The reasonable agreement between surrogate and CFD results, as summarized in the comparison table 8.1, validates the efficacy of the multi-agent workflow in identifying high-quality designs prior to expensive, physics-based simulations. We do note, however, that the epistemic uncertainty for the Bayesian model is high for

these final design candidates as compared to the original training dataset, leading to divergence between the surrogate and CFD results. This difference is particularly seen in  $C_M$  prediction, where the surrogate prediction shows a bias. We discuss potential mitigation strategies in the section 8.6.



**Figure 8.14:** Coefficient of pressure plots predicted by the agentic framework and the corresponding pressure distribution plot from OpenFoam simulation for the final four design candidates.

**Table 8.1:** Comparison between aerodynamic performance coefficient prediction using surrogate models at final design iteration against CFD predicted coefficients. We note that the surrogate model provides reasonable estimation of aerodynamic coefficients,  $C_D$  and  $C_L$  for these four design candidates. However, the epistemic uncertainty ( $\sigma$ ) when evaluating these final designs is high, as compared to the original training dataset. This is a consequence of design modifications performed on the airfoil shapes, which results in the CST parameters of the final candidates residing on the periphery of training data distribution leading to higher epistemic uncertainty. This bias is mitigated, to a certain extent, in the design selection strategy by choosing a composite utility score.

| Design ID        | Surrogate                   |                            |                            | CFD    |        |        |
|------------------|-----------------------------|----------------------------|----------------------------|--------|--------|--------|
|                  | $C_D$ ( $\sigma = 0.0011$ ) | $C_L$ ( $\sigma = 0.024$ ) | $C_M$ ( $\sigma = 0.012$ ) | $C_D$  | $C_L$  | $C_M$  |
| ID-151 (invalid) | 0.013                       | 0.836                      | -0.348                     | 0.016  | 0.88   | -0.145 |
| ID-394 (valid)   | 0.012                       | 0.761                      | -0.307                     | 0.0159 | 0.813  | -0.128 |
| ID-486 (invalid) | 0.01                        | 0.731                      | -0.283                     | 0.02   | 0.6682 | -0.087 |
| ID-970 (valid)   | 0.011                       | 0.741                      | -0.279                     | 0.018  | 0.713  | -0.097 |

Eventually, two design candidates: ID-394 and ID-970 are identified as potential solutions for this problem. Hereafter, these candidates can be used for detailed design that may include more advanced CFD simulation prior to design optimization and detailed design phase. We consider these steps as beyond the scope of our current work.

## 8.6 Limitations

While this work demonstrates a successful application of a multi-agent framework to a canonical engineering design problem, we acknowledge some limitations that offer avenues for future research. Firstly, the overall design workflow is orchestrated and supervised by a human manager. The agents function effectively as specialized assistants within a predefined structure, executing tasks such as design modification, sensitivity analysis, and performance vetting, but the high-level process logic and strategic decisions remain within the human domain. Consequently, this study does not demonstrate a fully autonomous, end-to-end design process, which would require agents capable of dynamic workflow planning and adaptation. A second limitation concerns the agents' ability to develop tools in-situ. We observed that the autonomous generation and integration of new analysis or utility scripts during the workflow was a challenge, often requiring numerous rounds of human-guided iteration to debug and rectify. The framework proved far more robust when supplied with a pre-vetted, comprehensive suite of tools prior to execution, indicating that while current LLM-based agents excel at tool use, reliable on-the-fly tool creation in complex engineering contexts remains an open research opportunity. Third, the scope of the CFD validation is constrained by the geometric complexity of the test case. Our automated CFD pipeline, particularly the meshing process which utilizes a structured C-grid topology, is tailored for 2D airfoil sections. Its direct applicability to more complex 3D geometries, such as wing-body junctions or internal turbine passages, which often necessitate advanced unstructured or hybrid meshing techniques, has not been explored. Some recent work such as [298] have explored using multi-agent setup for automating CFD flows, and could provide a future pathway to comprehensive design automation.

Finally, we note that the Bayesian surrogate used for predicting the aerodynamic coefficient can yield uncertain predictions when the airfoil design parameters reside on the periphery of the training data distribution. We observe this empirically since our

agent-based design selection strategy prioritized improvement in  $C_L$ , resulting in a bias in  $C_M$  prediction. Using the combined utility score helps reduce the downsides of this bias to certain extent, by assigning lower weight to  $U(C_M)$ . Another strategy can utilize the uncertainty estimate from the Bayesian model to determine whether the surrogate’s prediction is reliable or not. In particular, to mitigate this bias in  $C_M$  prediction for uncertain design samples, the framework could switch to using NeuralFoil surrogate, which provides reasonable estimates for  $C_M$ .

## 8.7 Summary

In this work, we build upon our prior research to introduce a multi-agent framework, guided by Large Language Models (LLMs), for assisting in the design and evaluation of engineering systems. We demonstrate this framework on the canonical problem of airfoil design, integrating it within a set-based design philosophy that incorporates formal risk management principles. The methodology employs a workflow managed by a human expert and executed by four specialized agents: a Coding Assistant, a Design Agent, a Systems Engineering Agent, and an Analyst Agent, each tasked with a specific function. Initially, the human Manager collaborates with the Coding Assistant to define the operational workflow and develop a suite of validated computational tools. Following this setup, the agents engage in a systematic process to progressively narrow a large set of potential design candidates, using the pre-developed tools and agent-based assessments to prune the design space.

A key contribution of our approach is the explicit integration of risk assessment into the automated design process. We adopt the Conditional Value-at-Risk (CVaR) as a risk metric to quantitatively filter out design candidates that exhibit a high probability of failing to meet the required design goal for the coefficient of lift ( $C_L$ ). The framework automates the initial, labor-intensive stages of design exploration by having the Analyst agent generate a global sensitivity analysis report. This report provides actionable heuristics

that guide the Design and Systems Engineering agents in the simultaneous analysis and modification of multiple designs. The process culminates in a human-in-the-loop stage where the human Manager acts as the final decision-maker. The framework augments the Manager's capability by presenting a curated final set of design candidates, along with high-fidelity Computational Fluid Dynamics (CFD) simulation results, to inform the ultimate selection. Looking forward, future work will focus on two primary directions: first, enhancing agent autonomy to enable dynamic workflow adaptation in response to real-time findings, and second, extending the framework's application to more complex 3D multi-physics problems, which would necessitate direct integration with industry-standard Computer-Aided Design (CAD) platforms.



# CHAPTER 9

## Summary and Future Work

### 9.1 Summary

This thesis has tried to address two fundamental bottlenecks in the modern engineering design process: the prohibitive cost of high-fidelity physical simulations and the intricate, knowledge-intensive nature of the design workflow itself. The research presented herein is organized into two interconnected parts. The first part focuses on developing geometry-aware neural operators, capable of delivering rapid and accurate physical predictions. The second part focuses on ways to improve the design process, detailing the development of Large Language Model (LLM) powered multi-agent frameworks that can automate, augment, and reason through complex engineering design tasks. Together, these two research ideas can be used towards a more efficient, intelligent, and ultimately autonomous engineering design paradigm.

Part I of this thesis addresses a research gap in traditional neural operators: their inability to generalize across varying geometric domains. For surrogate models to be truly effective in design optimization, where an object’s shape is the primary variable, they must be inherently geometry-aware. Our work progressively builds this capability through a series of novel neural operator frameworks. We established the Multi-Task DeepONet (MT-DeepONet), a framework designed for synergistic learning across multiple distinct

geometries and PDE parameters within a single training session. By encoding geometric information via a binary mask, this approach demonstrated significant gains in training efficiency and transfer learning capabilities for problems in Darcy flow and heat transfer. We then introduced Fusion-DeepONet to tackle the dual challenges of data scarcity and complex physics prevalent in aerospace engineering. This highly data-efficient operator proved robust in learning hypersonic and supersonic flows over varying geometries on both structured and unstructured grids, a crucial feature for real-world applications. Its utility was further enhanced by a derivative-enhanced loss term, enabling accurate prediction of critical surface quantities like heat flux. Finally, we showcased the versatility of our operator-learning approach by developing a hybrid DeepONet-Transolver framework to analyze the complex, nonlinear buckling behavior of PET bottles. This work demonstrated the ability to handle intricate, non-parametric geometric variations and perform multi-task predictions, simultaneously learning full-field nodal displacements and time-dependent reaction forces. Collectively, the contributions in Part I deliver a set of geometry-aware surrogate models applicable across fluid and solid mechanics, laying the groundwork for rapid, in-the-loop design evaluation.

Part II addresses the broader context in which simulation tools are used: the engineering design process. This process is a fundamentally collaborative and cognitive endeavor that has remained largely beyond the scope of automation. To address this, we harnessed the reasoning and language capabilities of modern LLMs to structure and streamline this workflow. Our initial exploration, MyCrunchGPT, demonstrated the feasibility of using a single LLM as a high-level conductor to orchestrate the entire scientific machine learning pipeline from a user’s natural language prompts. We then evolved this concept into a more sophisticated multi-agent paradigm that better mirrors human engineering teams. We introduced a foundational framework of specialized AI agents, such as a Design Engineer and a Systems Engineer, that engage in iterative design-and-review loops. A key innovation was grounding these agents in verifiable domain knowledge by using a Graph Ontologist agent to construct domain-specific knowledge graphs from technical literature,

providing a structured and reliable external memory. Our final contribution extended this framework to the challenging early stages of design, which are characterized by vast parameter spaces and high uncertainty. By integrating a set-based design philosophy and principles of formal risk management, we developed a system where a team of agents, overseen by a human manager, systematically explores and prunes the design space. This framework uses quantitative risk metrics like the Conditional Value-at-Risk (CVaR) to filter out unpromising candidates, automating the laborious initial exploration and delivering a curated, risk-assessed set of designs to the human expert. In essence, Part II presents a novel pathway toward autonomous systems that can effectively partner with human engineers, enhancing the efficiency, creativity, and robustness of the design process.

## 9.2 Future Work

While this thesis establishes a strong foundation for AI-driven simulation and design, the journey toward truly autonomous and integrated engineering systems presents several future research opportunities. The following directions build directly upon the work presented, aiming to enhance the capability of individual components and, more importantly, to forge a synergy between them.

First, there is a continuing need to advance the accuracy and robustness of the neural surrogate models developed in Part I. The hybrid DeepONet-Transolver framework, while successful in capturing the complex buckling behavior of PET bottles, can be further improved. Future work can focus on architectural refinements and the incorporation of more nuanced physics-informed constraints to enhance its predictive accuracy for both displacement fields and reaction forces, particularly in highly localized regions of deformation. A critical aspect of this research will involve a rigorous benchmarking of the improved framework against a new generation of neural operator and geometric deep learning formulations that have recently been proposed, thereby establishing its competitive standing.

Second, a significant leap in the capability of the multi-agent frameworks from Part II involves empowering them with more direct and sophisticated tools for design generation and analysis. Currently, the agents operate on parameterized representations of geometry. A transformative next step is to develop a Design Agent that can interact directly with Computer-Aided Design (CAD) software through its APIs. This would enable the agent to perform complex geometric manipulations and generative design tasks that are not possible with simple parameter tuning. Complementing this, a comprehensive Simulation Agent could be developed to autonomously manage the end-to-end simulation pipeline. This agent would receive a CAD model from the Design Agent and handle the entire workflow of model preparation, physics setup, and, most critically, automated meshing. Overcoming the challenge of robust and automated meshing for arbitrary and evolving geometries is essential for creating a truly autonomous design-and-analysis loop.

A third and more ambitious direction is to scale the multi-agent paradigm to address large-scale, multidisciplinary system design problems. The current framework excels at component-level design, such as an airfoil. The next frontier is system-of-systems design, such as an entire aircraft or vehicle, where teams of engineers from different domains (e.g., aerodynamics, structures, propulsion, controls) must collaborate. Future research could explore a hierarchical multi-agent architecture where specialized teams of agents, each focused on a specific subdomain, work in parallel. These teams would need to communicate, negotiate trade-offs, and resolve conflicts under the coordination of a higher-level Chief Engineer agent responsible for maintaining system-level integrity and meeting overarching performance targets. This would require developing advanced protocols for multi-agent collaboration and distributed problem-solving, pushing the boundaries of AI to mirror the complex organizational dynamics of large engineering projects.

Ultimately, these avenues of future work point toward a unified vision where the AI ‘engineers’ developed in Part II are equipped with the advanced AI ‘simulators’ from Part I. Realizing these ideas can help accelerate the pace of innovation and enable the design of more complex and optimized systems than ever before.

# Appendices

## A1 Appendices for Chapter 2

### A1.1 Empirical results for two-branch framework (MIONet)

In this section, we discuss briefly our empirical findings with the MIONet framework which can be used alternatively for solving the Fisher equation problem proposed in this study. The two associated tasks for this problem, mapping random initial conditions ( $\mathbf{u}(x, t = 0)$ ) and equation parameters  $(\alpha, \beta, \gamma, \delta)$  to solution field  $u(x, t)$  can be provided as inputs to two separate branch networks that are combined using element-wise product before combining with the trunk network embeddings. This mapping is defined as

$$\mathcal{G}_{\theta}((\alpha, \beta, \gamma, \delta), \mathbf{u}(x, t = 0))(\zeta) := \sum_{i=1}^p \left[ b_i^{(1)}(\alpha, \beta, \gamma, \delta) \odot b_i^{(2)}(\mathbf{u}(x, t = 0)) \right] \odot tr_i(\zeta), \quad (\text{A1.1})$$

where  $b^{(k)}, k = 1, 2$  represents the outputs of the two branch networks and  $\odot$  denotes the element-wise product for their embeddings. We train this network using Adam optimizer with an exponentially decaying learning rate with an initial value of  $1 \times e^{-3}$  for 50,000 epochs. Details for the MIONet framework can be found in Table A2. The relative  $\mathcal{L}_2$  norm observed for the 800 test samples is  $\approx 5.1\%$ , which is marginally higher as compared to the proposed MT-DeepONet framework ( $\approx 2.4\%$ ). Additionally, the MIONet framework uses a total of 159,712 trainable parameters as compared to the MT-DeepONet which uses 136,664. Thus, we observe that MT-DeepONet yields marginally better results than the MIONet framework in terms of computational cost and accuracy.

### A1.2 Data Generation

In this section, we present relevant details related to the data generation process for the three problems investigated in this study.

## Fisher equations

For this problem, the goal is to learn the operator mapping from the random initial density condition  $u$  across a parameter range  $a$  and  $b$ , for its entire time evolution. This mapping is expressed as  $\mathcal{G} : u(\mathbf{x}, a, b, t = 0) \mapsto u(\mathbf{x}, t)$ , where  $t > 0$  and  $\mathbf{x} \times t \in [0 \times 1] \times [0 \times 1]$ . The initial density is modeled as a Gaussian random field (GRF), defined by:

$$u(\mathbf{x}, t = 0) \sim \mathcal{GP}(u(\mathbf{x}, t = 0) \mid \mu(\mathbf{x}), \text{Cov}(\mathbf{x}, \mathbf{x}')), \quad (\text{A1.2})$$

where  $\mu(\mathbf{x})$  and  $\text{Cov}(\mathbf{x}, \mathbf{x}')$  are the mean and covariance functions, respectively. We set  $\mu(\mathbf{x}) = 5 + 0.1 \times \sin(\pi x)$ , while the covariance matrix is defined by the squared exponential kernel:

$$\text{Cov}(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|_2^2}{\ell^2}\right), \quad (\text{A1.3})$$

where  $\ell = 0.4$  is the correlation length, and  $\sigma^2 = 2$  is the variance. We utilize Karhunen-Lo  ve expansion (KLE) to generate 1,000 random initial conditions. The temporal points  $t$  and spatial points  $\mathbf{x}$  are discretized into a  $20 \times 64$  grid, resulting in a total of 1,280 collocation points. Parameters  $a$  and  $b$  are discretized in steps of 0.1 and 0.15, respectively, within the defined parameter space. Combining different parameter values for  $a$  and  $b$  with the random initial conditions, we generate a total of 4,000 training samples. This dataset is split into training and testing sets using an 80% – 20% split, with  $N_{\text{train}} = 3,200$  and  $N_{\text{test}} = 800$ . For preparing data for the multi-task operator network, we concatenate the initial condition data at 64 domain points with the parameter values for the three Fisher equations, resulting in a 68 dimensional input vector (64 initial condition points + 4 equation parameters).

## Darcy equation with transfer learning across multiple geometries

The multi-task objective in this problem is to learn the operator for the Darcy flow equation over different 2D spatial domains under randomly generated initial states for

conductivity field  $K(\mathbf{x})$ . The hydraulic conductivity field is modeled as a stochastic process, using truncated Karhunen-Loève expansion. The conductivity field is generated in a reference square domain  $\Omega \in [0, 1] \times [0, 1]$  with 100 grid points using a Gaussian random field with length scale,  $\ell = 0.05$ . A total of 6000 solution fields are generated using the multiple initial values of conductivity fields for each geometry. A Dirichlet boundary condition of  $h(\mathbf{x}) = 0 \quad \forall \mathbf{x} \in \partial\Omega$  is applied on all the edges of each geometry. The solution to Darcy’s equation for all samples is generated using MATLAB’s PDE Toolbox solver. The domain mesh is generated using the MATLAB PDE Toolbox mesh generation algorithm with a maximum mesh size of 0.03 inches for all geometries. We use unstructured triangular elements for the mesh. The hydraulic pressure head solution obtained is linearly interpolated on a regular grid in  $\Omega \in [0, 1] \times [0, 1]$  to utilize a common trunk network input. Points in the square grid  $\Omega$  that lie outside the triangular geometry are extracted and the solution field at such points is manually set to 0 throughout the dataset. We sub-divide the 6000 samples into  $N_{train} = 5400$  and  $N_{test} = 600$  for each source model. For training the MT-DeepONet with multiple geometries, we use an equal number of samples from each geometry to obtain a combined total of  $N_{train} = 5400$ , thus ensuring equal representation.

### Heat transfer equation with multiple 3D geometry

To generate data for the steady-state heat transfer problem with multiple plate geometries, we create a parametric CAD model of the plate where the parameters  $\mathbf{n}$  and  $\mathbf{d}$  determine the number of holes and their location relative to the central axis of the plate protrusion, respectively. To generate unique geometries with a combination of  $n$  and  $d$ , we fix the location of one hole across all the geometries. We solve the steady-state heat transfer using MATLAB’s PDE Toolbox for various design configurations that result from varying the design parameters  $\mathbf{n}$  and  $\mathbf{d}$  in the plate geometry, where  $2 \leq \mathbf{n} \leq 9$  and  $0.9 \leq \mathbf{d} \leq 1.6$ . We increment  $\mathbf{d}$  in steps of 0.1 resulting in a total of 64 distinct design configurations. As discussed earlier, we separate the training and test cases based on the



hole location,  $\mathbf{d}$  as shown in Table A1. For each design, we use PDEToolbox for mesh generation with a maximum element size of 0.1 inch and tetrahedral elements. A heat source  $\mathbf{q} = 1$  is placed inside each hole, simulating a cartridge heater to heat the plate. Convective boundary conditions are applied on the protruding face and side walls with an ambient temperature  $u_\infty = 6$  and a convective heat transfer coefficient  $h_c = 0.3$ . The output solution, which is the temperature field  $u$ , is evaluated at all node points of the mesh. The temperature solutions are initially obtained on unstructured grid points. For simplifying the inputs for the trunk network, we interpolate these solutions over a regular 3D grid ( $100 \times 100 \times 100$ ) around the plate. Points in the regular grid that fall outside the domain of the 3D plate are set to 0 before training the network.

**Table A1:** *Training and test sample selection for 3D plate problem. We select specific radial locations ( $d$ ) of the holes for training to encompass the innermost and outermost locations as well as the median location. We interpolate the solution on other radial locations during model inference.*

| $\mathbf{n}$ | $\mathbf{d}$<br>(training samples) | $\mathbf{d}$<br>(testing samples) |
|--------------|------------------------------------|-----------------------------------|
| 2            | {0.9, 1.2, 1.6}                    | {1.0, 1.1, 1.3, 1.4, 1.5}         |
| 3            | {0.9, 1.2, 1.6}                    | {1.0, 1.1, 1.3, 1.4, 1.5}         |
| 4            | {0.9, 1.2, 1.6}                    | {1.0, 1.1, 1.3, 1.4, 1.5}         |
| 5            | {0.9, 1.2, 1.6}                    | {1.0, 1.1, 1.3, 1.4, 1.5}         |
| 6            | {0.9, 1.2, 1.6}                    | {1.0, 1.1, 1.3, 1.4, 1.5}         |
| 7            | {0.9, 1.2, 1.6}                    | {1.0, 1.1, 1.3, 1.4, 1.5}         |
| 8            | {0.9, 1.2, 1.6}                    | {1.0, 1.1, 1.3, 1.4, 1.5}         |
| 9            | {0.9, 1.2, 1.6}                    | {1.0, 1.1, 1.3, 1.4, 1.5}         |

### A1.3 Data pre-processing

The multi-task operator network design for all the four problems discussed earlier consists of a branch and trunk network, where the branch creates a map between the varying input functions (equation representations, geometry, initial conditions) and the

output function. The inputs to the branch network are formed by combining the feature representations of the input functions. The input to the trunk network for the Fisher problem consists of spatio-temporal locations  $\mathbf{x} \times \mathbf{t} \in \mathbb{R}^n \times \mathbb{R}$ , where  $n = 1, 2$ . For the Darcy and 3D heat transfer problem, the trunk input consists only of spatial points  $\mathbf{x} \in \mathbb{R}^n$  where  $n = 2, 3$ . To improve network training, the available data is scaled to reduce to the same order. In particular, the input initial condition and outputs for Darcy, and Fisher problems are scaled using the mean and standard deviation of the training dataset using the operation:

$$x_{in} = \frac{x - \mu}{\sigma} \quad (\text{A1.4})$$

For the 3D heat transfer problem, we normalize the inputs and outputs using min-max scaling given by:

$$x_{in} = \frac{x - x_{min}}{x_{max} - x_{min}} \quad (\text{A1.5})$$

where  $x_{max}$  and  $x_{min}$  represent the minimum and maximum values across the sample set.

## A1.4 Network Architecture

Our network design and hyper-parameter selection are determined by the problem under investigation. In Table A2, we provide a summary of the network architecture and hyper-parameters used for different problems in this study.

**Table A2:** *Details of network architecture used for different multi-task problems.*

| Problem              | Branch network |                                    | Trunk network<br>(neurons per layer)        | Activation        | Regularizer     | Dropout<br>(branch) | Masking |
|----------------------|----------------|------------------------------------|---------------------------------------------|-------------------|-----------------|---------------------|---------|
|                      | Type           | Neurons per layer                  |                                             |                   |                 |                     |         |
| Fisher (MT-DeepONet) | MLP            | [68, 128, 128, 300]                | [2, 128, 128, 128, 300]                     | Leaky_ReLU        | $\mathcal{L}_2$ | None                | None    |
| Fisher (MIONet)      | MLP            | Br(1) : [4, 32, 64, 128, 128, 128] | [2, 128, 128, 128, 128, 128]                | Br: Tanh,         | $\mathcal{L}_2$ | None                | None    |
|                      | MLP            | Br(2): [64, 64, 64, 128, 128, 128] |                                             | Trunk: Leaky_ReLU |                 |                     |         |
| Darcy                | 2D CNN         | CNN filters: [16, 32, 64, 64]      | [2, 64, 128 $\times$ 3, 150]                | Conv: Tanh, ReLU  | None            | 0.1                 | Yes     |
|                      | MLP            | MLP: [128, 128, 150]               |                                             | MLP: Leaky_ReLU   |                 |                     |         |
| 3D heat transfer     | MLP            | [2, 32, 64, 128, 128, 200]         | [3, 32, 64 $\times$ 3, 128 $\times$ 5, 200] | swish             | None            | 0.1                 | Yes     |

## A2 Appendices for Chapter 3

### A2.1 Datasets

In this section, we report on the training and testing samples for the three problems reported in this study. Table A3 presents the values of the input parameters for the test and training data sets of the semi-ellipse problem. Table A4 demonstrates the values of geometric and flow conditions input parameters for the reentry capsule problem used for training and testing. In Table A5, we report the input geometric parameters of the converging-diverging nozzle selected for training and testing neural operators.

**Table A3:** *Training and testing samples for input parameters of surrogate models of inviscid hypersonic flow over semi-elliptic blunt bodies.*

| Case | $(a, b)$ Train | Case | $(a, b)$ Train | Case | $(a, b)$ Train | Case | $(a, b)$ Train | case | $(a, b)$ Test |
|------|----------------|------|----------------|------|----------------|------|----------------|------|---------------|
| 1    | (2.78,0.70)    | 9    | (2.89,1.26)    | 17   | (1.27,1.30)    | 25   | (2.36,1.09)    | 1    | (0.90,0.80)   |
| 2    | (1.05,0.88)    | 10   | (1.51,1.69)    | 18   | (0.53,0.53)    | 26   | (1.62,0.84)    | 2    | (2.64,1.36)   |
| 3    | (1.57,1.03)    | 11   | (2.08,0.74)    | 19   | (2.72,1.53)    | 27   | (1.21,1.44)    | 3    | (1.44,1.16)   |
| 4    | (0.77,1.66)    | 12   | (1.90,1.39)    | 20   | (2.46,1.00)    | 28   | (1.79,1.40)    | 4    | (0.80,0.94)   |
| 5    | (2.27,0.64)    | 13   | (2.18,1.75)    | 21   | (1.69,1.78)    |      |                | 5    | (2.41,1.25)   |
| 6    | (0.66,1.20)    | 14   | (2.56,1.59)    | 22   | (2.84,1.08)    |      |                | 6    | (0.98,1.63)   |
| 7    | (2.99,0.90)    | 15   | (1.10,1.50)    | 23   | (1.35,0.59)    |      |                | 7    | (2.10,0.76)   |
| 8    | (0.63,1.15)    | 16   | (1.15,1.55)    | 24   | (1.85,0.54)    |      |                | 8    | (2.00,0.68)   |

**Table A4:** *Training and testing samples for input parameters of surrogate models for reentry capsule.*

| Case | $(R, \alpha, Ma)$ Train | Case | $(R, \alpha, Ma)$ Train | Case | $(R, \alpha, Ma)$ Train | Case | $(R, \alpha, Ma)$ Train | Case | $(R, \alpha, Ma)$ Train | Case | $(R, \alpha, Ma)$ Train | Case | $(R, \alpha, Ma)$ Test | Case | $(R, \alpha, Ma)$ Test |
|------|-------------------------|------|-------------------------|------|-------------------------|------|-------------------------|------|-------------------------|------|-------------------------|------|------------------------|------|------------------------|
| 1    | (19.05,50.56,6.97)      | 11   | (16.59,57.58,5.00)      | 21   | (13.47,41.35,6.42)      | 31   | (18.80,45.39,6.36)      | 41   | (14.19,51.65,5.67)      | 1    | (15.79,53.26,7.78)      | 11   | (17.25,54.19,7.36)     |      |                        |
| 2    | (18.61,42.64,6.06)      | 12   | (15.61,56.70,5.27)      | 22   | (13.37,40.35,5.14)      | 32   | (19.24,48.12,8.00)      | 42   | (14.56,43.68,6.13)      | 2    | (14.89,43.10,6.33)      | 12   | (16.92,56.39,7.68)     |      |                        |
| 3    | (16.46,51.94,7.40)      | 13   | (13.08,52.46,5.31)      | 23   | (17.60,58.94,7.23)      | 33   | (19.76,53.95,7.52)      | 43   | (18.12,47.01,7.56)      | 3    | (18.29,44.97,7.28)      |      |                        |      |                        |
| 4    | (13.80,52.22,5.72)      | 14   | (19.50,41.78,5.56)      | 24   | (14.70,59.20,6.74)      | 34   | (14.94,59.68,7.13)      | 44   | (14.37,46.97,7.00)      | 4    | (14.59,44.40,6.24)      |      |                        |      |                        |
| 5    | (19.93,45.29,5.79)      | 15   | (19.81,47.85,5.21)      | 25   | (19.60,42.95,7.47)      | 35   | (16.84,52.97,7.34)      | 45   | (18.70,56.24,6.62)      | 5    | (17.87,49.40,5.54)      |      |                        |      |                        |
| 6    | (17.14,57.07,5.83)      | 16   | (17.37,46.63,5.45)      | 26   | (17.76,55.01,6.26)      | 36   | (16.66,50.92,7.85)      | 46   | (17.53,55.51,6.04)      | 6    | (16.18,40.07,5.97)      |      |                        |      |                        |
| 7    | (13.21,40.90,6.66)      | 17   | (13.59,49.99,5.88)      | 27   | (18.00,48.95,6.49)      | 37   | (19.18,58.51,5.64)      | 47   | (15.17,54.85,5.09)      | 7    | (15.54,44.30,6.77)      |      |                        |      |                        |
| 8    | (16.35,47.57,6.60)      | 18   | (15.08,49.07,7.87)      | 28   | (18.88,45.86,6.92)      | 38   | (16.06,46.03,7.63)      | 48   | (13.97,50.32,7.73)      | 8    | (14.02,48.62,7.19)      |      |                        |      |                        |
| 9    | (18.44,42.24,7.93)      | 19   | (14.29,57.90,5.42)      | 29   | (17.09,54.38,7.06)      | 39   | (15.84,55.70,6.16)      |      |                         | 9    | (19.33,53.46,5.37)      |      |                        |      |                        |
| 10   | (15.35,58.15,6.85)      | 20   | (13.70,51.21,6.50)      | 30   | (16.22,41.18,5.17)      | 40   | (15.39,43.37,6.89)      |      |                         | 10   | (18.20,59.52,5.95)      |      |                        |      |                        |

**Table A5:** Training and testing samples for input parameters of surrogate models for supersonic flow in the converging diverging nozzle.

| Case | $(h_i, h_o, x_t)$ Train | Case | $(h_i, h_o, x_t)$ Train | Case | $(h_i, h_o, x_t)$ Train | Case | $(h_i, h_o, x_t)$ Train | Case | $(h_i, h_o, x_t)$ Train | Case | $(h_i, h_o, x_t)$ Test | Case | $(h_i, h_o, x_t)$ Test |
|------|-------------------------|------|-------------------------|------|-------------------------|------|-------------------------|------|-------------------------|------|------------------------|------|------------------------|
| 1    | (3.86,2.03,0.58)        | 10   | (3.33,2.41,0.57)        | 19   | (3.18,2.39,0.48)        | 28   | (3.84,1.79,0.58)        | 37   | (3.88,2.43,0.49)        | 1    | (3.73,1.85,0.62)       | 10   | (3.22,1.72,0.53)       |
| 2    | (3.80,1.63,0.52)        | 11   | (3.51,2.38,0.45)        | 20   | (3.09,2.06,0.55)        | 29   | (3.58,2.22,0.59)        | 38   | (3.43,1.80,0.63)        | 2    | (3.19,1.85,0.58)       | 11   | (3.69,1.97,0.49)       |
| 3    | (3.49,2.10,0.61)        | 12   | (3.37,2.33,0.47)        | 21   | (3.06,1.57,0.54)        | 30   | (3.46,1.56,0.46)        | 39   | (3.40,2.29,0.53)        | 3    | (3.81,2.31,0.56)       | 12   | (3.45,1.50,0.51)       |
| 4    | (3.11,2.11,0.50)        | 13   | (3.00,2.12,0.47)        | 22   | (3.04,1.52,0.46)        | 31   | (3.83,1.77,0.54)        | 40   | (3.34,1.67,0.58)        | 4    | (3.64,2.28,0.52)       | 13   | (3.36,1.71,0.57)       |
| 5    | (3.99,1.76,0.50)        | 14   | (3.93,1.59,0.49)        | 23   | (3.65,2.45,0.60)        | 32   | (3.89,1.91,0.65)        | 41   | (3.16,2.08,0.49)        | 5    | (3.30,2.24,0.46)       | 14   | (3.14,1.93,0.60)       |
| 6    | (3.59,2.35,0.51)        | 15   | (3.97,1.89,0.46)        | 24   | (3.24,2.46,0.57)        | 33   | (3.97,2.20,0.62)        | 42   | (3.22,1.68,0.53)        | 6    | (3.13,2.02,0.63)       | 15   | (3.90,2.17,0.47)       |
| 7    | (3.02,1.54,0.56)        | 16   | (3.62,1.83,0.48)        | 25   | (3.94,1.65,0.61)        | 34   | (3.27,2.48,0.59)        |      |                         | 7    | (3.39,2.16,0.64)       | 16   | (3.74,2.48,0.51)       |
| 8    | (3.47,1.88,0.56)        | 17   | (3.08,2.00,0.51)        | 26   | (3.68,2.25,0.53)        | 35   | (3.54,2.15,0.61)        |      |                         | 8    | (3.26,1.66,0.54)       | 17   | (3.60,2.21,0.61)       |
| 9    | (3.78,1.61,0.65)        | 18   | (3.29,1.95,0.64)        | 27   | (3.71,1.95,0.55)        | 36   | (3.52,2.05,0.64)        |      |                         | 9    | (3.75,1.75,0.60)       | 18   | (3.56,2.32,0.63)       |

## A2.2 High pressure ratio Sod problem (LeBlanc problem)

For completeness, we evaluated the Fusion-DeepONet on the challenging high-pressure-ratio Sod problem, also known as the LeBlanc problem. This problem is complicated due to the steep pressure discontinuity it involves. Recent research has explored various neural operators designed to handle sharp discontinuities in this context. For example, Peyvan *et al.* [8] and Shih *et al.* [9] utilized approaches such as parameter-conditioned UNet, two-step Rowdy DeepONet, and Transformers to tackle the high-pressure-ratio Sod problem effectively.

We solve the 1D compressible Euler equations over the spatial domain  $x \in [-20, 20]$ . The initial conditions for this problem are defined as:

$$(\rho, u, p) = \begin{cases} (2.0, 0.0, p_l) & x \leq -10 \\ (0.001, 0.0, 1.0) & x > -10, \end{cases} \quad (\text{A2.1})$$

where  $p_l \in [10^9, 10^{10}]$ .

We use the exact solution method to compute results at  $t_f = 0.0001$ . 500 equispaced  $p_l$  values are sampled, with 400 cases randomly assigned for training and 100 for testing. We construct the loss function to train the neural networks using the logarithm of density and pressure values. We want to create a mapping from  $p_l$  to the final solution for  $\rho$ ,  $u$  and  $p$ . During inference, the exponential function is applied to the predicted logarithmic values of density and pressure, converting them back to their physical values. For Fusion-DeepONet, we chose three hidden layers, each with 100 neurons in the branch and trunk networks. We

**Table A6:** Comparison of  $L_2$  norms of the error for all the unseen test cases of the LeBlanc problem predicted by the trained 2-step DeepONet [8], UNet [8], Transformer [9], and Fusion-DeepONet.

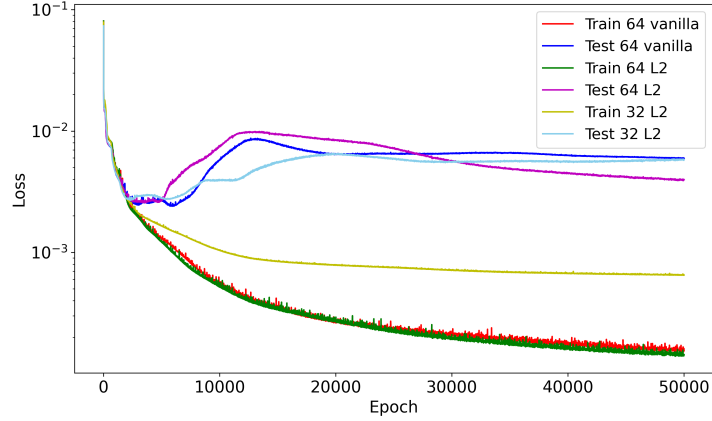
| Framework           | $\%L_2(\rho)$ | $\%L_2(u)$  | $\%L_2(p)$  | $\%L_2$     |
|---------------------|---------------|-------------|-------------|-------------|
| 2 step DeepONet [8] | 0.66          | 3.39        | 2.86        | 2.31        |
| UNet[8]             | 1.00          | 2.65        | 2.27        | 1.97        |
| Transformer [9]     | 1.42          | 2.27        | 2.23        | 1.97        |
| Fusion-DeepONet     | <b>0.61</b>   | <b>0.59</b> | <b>1.23</b> | <b>0.81</b> |

employed a single trunk network to predict all three variables, including  $\rho$ ,  $u$ , and  $p$ . We then trained the network for 100,000 epochs with an exponential decay profile for learning rate. The exponential decay setup is identical to the one we used for the semi-ellipse geometry-dependent problem. Table. A6 demonstrates a comparison of Fusion-DeepONet accuracy versus 2-step DeepONet [8], UNet [8], and Transformer [9] in predicting the discontinuous solution of the LeBlanc problem. According to Table. A6, Fusion-DeepONet performs better than all its counterparts in predicting all the flow variables. This test showcases the generalization of the Fusion-DeepONet prediction in learning discontinuous solutions.

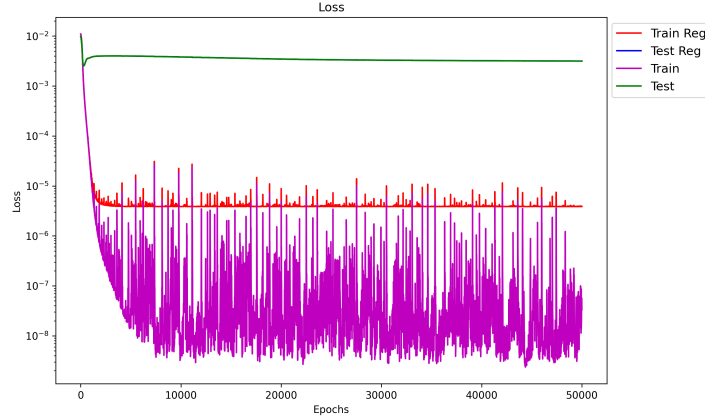
### A2.3 Vanilla- and POD-DeepONet Hyperparameter Tuning

We employed three or five hidden layers, each with 32 or 64 neurons. We also applied L1 and L2 regularizers for the weights of the branch subnetwork to avoid overfitting in the Vanilla DeepONet. FigureA1 compares the training and testing MSE for the L2 regularizer to train a Vanilla DeepONet with three hidden layers in the branch and trunk subnetworks. The width of all the hidden layers is either 32 or 64. According to Fig. A1, the L2 regularizer reduces the generalization error; however, the error is much higher than that of Fusion-DeepONet. We tested the L1 regularizer and adjusted the model’s size, but since the problem of semi-ellipses is very challenging, Vanilla-DeepONet, with various

hyperparameters and regularizers, is unable to generalize well on this shape-dependent hypersonic flow problem. For completeness, we show the MSE trend of POD-DeepONet during training in Fig. A2. We applied a coefficient of  $10^{-2}$  for the L2 regularizer. Even with the regularizer, the POD-DeepONet is unable to learn the semi-ellipse problem.



**Figure A1:** Comparison of MSE for training vanilla DeepONet with various model sizes and L2 regularizer. Vanilla DeepONet is trained for the inviscid hypersonic flow over semi-ellipse problem on uniform grid. The term Vanilla indicates no regularizer and the number 32 or 64 in the legend refers to the number of neurons used in each hidden layer of trunk and branch networks.



**Figure A2:** Comparison of regularized POD-DeepONet and not regularized POD-DeepONet MSE values versus epochs. The term "reg" indicates the use of L2 regularizer.

## A2.4 Neural Operators

This section describes the architectures of various neural operators used to learn the hypersonic flow field around blunt bodies and supersonic flow in converging-diverging

nozzle. Our goal is to learn the flow by using the object’s geometry as input.

## Vanilla-DeepONet

Lu *et al.* [299] developed deep operator networks (DeepONet) inspired by the universal approximation theorem of operators to map from an infinite dimension functional input to another infinite dimension functional output, both defined in the Banach space. The DeepONet framework consists of two distinct networks, namely branch and trunk nets. The branch net encodes the input parameters, and the trunk net encodes the independent variables, namely spatial coordinates and time. For this study, we focus on learning the solution at a specific time and avoiding learning the time-dependent dynamic. Therefore, we only input spatial coordinates into the trunk net and geometric parameters into the branch network. We multiply the trunk net output by the branch net output to construct the final prediction of DeepONet. The output of the DeepONet consists of four flow variables: density, x-velocity, y-velocity, and pressure. With DeepONet, we try to learn an operator  $\mathcal{G}$  that

$$\mathcal{G} : h_g \mapsto \mathbf{U}(x, y) \quad (\text{A2.2})$$

The operator  $\mathcal{G}$  maps from geometric parameters denoted as  $h_g$  to the solution field  $\mathbf{U} = (\rho(x, y), u(x, y), v(x, y), p(x, y))^T$  within a physical domain described in Eq. (3.2.2). The DeepONet framework can be described using a compact form as

$$\mathbf{U}_s(x, y) = \sum_{k=0}^{l_d} B_{k,s}(h_g; \theta) T_k(x, y; \Gamma) \quad (\text{A2.3})$$

where  $B_{k,s}$  denotes the branch network for the  $s^{\text{th}}$  output variable and at  $k^{\text{th}}$  latent dimension. Let  $HB$  be the total number of hidden layers in the branch network and  $HT$  be the total number of hidden layers in the trunk network. We can describe the branch and trunk outputs as

$$B_{k,s} = W_{ij}^{b,o} \mathbf{z}_j^{HB} + b_i^{b,o}, \quad (\text{A2.4})$$

and

$$T_k = W_{ij}^{t,o} \mathbf{a}_j^{HT} + b_i^{t,o}, \quad (\text{A2.5})$$

where  $\mathbf{z}_j^{HB}$  and  $\mathbf{a}_j^{TB}$  denote the outputs of the last hidden layer of branch and trunk networks, respectively. The outputs of the  $l^{\text{th}}$  hidden layer of the branch and trunk networks can be formulated as

$$\mathbf{z}^l = W_{ij}^{b,l} \sigma(\mathbf{z}_j^{l-1}) + b_i^{b,l} \quad (\text{A2.6})$$

$$\mathbf{a}^l = W_{ij}^{t,l} \sigma(\mathbf{a}_j^{l-1}) + b_i^{t,l} \quad (\text{A2.7})$$

and then for the input layer ( $l = 1$ ) of the branch and trunk network, we have

$$\mathbf{z}_j^0 = \mathbf{H}_j, \quad j = 1, \dots, p$$

$$\mathbf{a}_j^0 = \mathbf{X}_j, \quad j = 1, 2$$

where  $\mathbf{H}_j \in \mathbb{R}^{N_s}$  and  $\mathbf{X}_j \in \mathbb{R}^{N_p}$  indicate the input vectors of branch and trunk networks for the  $j^{\text{th}}$  parameter and  $j^{\text{th}}$  coordinate, respectively. In Eq. (A2.7),  $\sigma$  denotes the activation function and superscript  $b$ , and  $t$  refers to the quantity assigned to the branch and trunk networks.

## POD-DeepONet

Lu *et al.* [27] introduced POD-DeepONet that replaces trunk net with the training data's proper orthogonal decomposition (POD) modes. POD-DeepONet employs only a fully connected network for the branch net that learns the coefficients of the POD basis to construct the predictions. For POD-DeepONet, the predictions are approximated using



the following formula:

$$\mathbf{U}_s(x, y) = \sum_{k=0}^{l_d} B_{k,s}(h_g; \theta) T_k(\xi) + T_0(\xi) \quad (\text{A2.8})$$

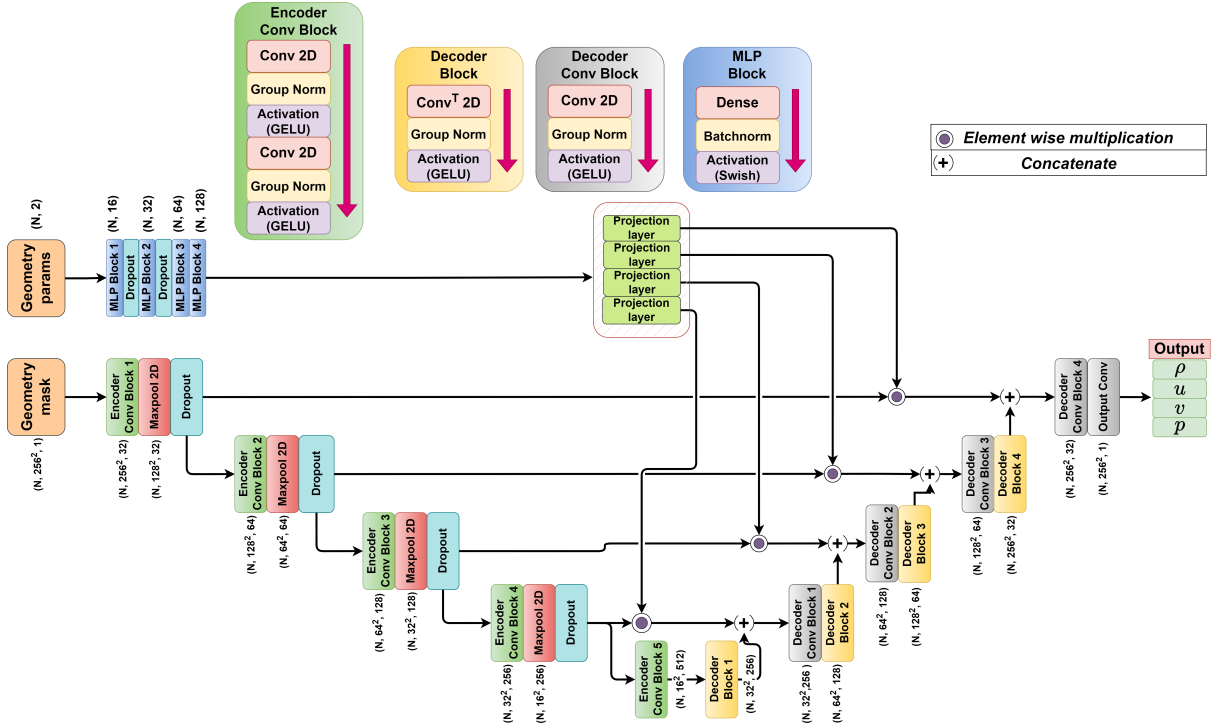
where  $T_0(\xi)$  is the mean function of training target functions such as  $\rho$ ,  $u$ ,  $v$ , and  $p$  variables.  $T_k(\xi)$  are the POD basis functions computed using the target functions after removing the mean values.

### U-Net with parameter conditioning for uniform grid

U-Net, introduced by Ronneberger [300], was developed as an image segmentation convolution network. Later, [301] introduced a modified version of U-Net with parameter conditioning for generalizing the solution to PDEs across multiple conditions using the same network. For learning hypersonic flow field across multiple elliptical shapes, we utilize the parameter conditioning of U-Net and use a framework similar to [301] consisting of two network types: a multi-layer perceptron network (MLP) and a convolution encoder-decoder network. See Figure A3 showing network details. Each MLP block consists of a linear layer followed by *Batchnorm* and *Activation* layers. We use ‘swish’ activation function defined as  $\sigma = \mathbf{x} \cdot \text{sigmoid}(\mathbf{x})$ . The U-Net encoder and decoder modules contain a series of Encoder and Decoder blocks. Each *Encoder* block consists of two sequential convolution layers, each followed by a *Groupnorm* and *Activation* layers. The *Decoder* modules consist of two types of blocks: a *Decoder* block and a *Decoder Convolution* block. The *Decoder* block consists of transposed convolution layers that upsample the image embeddings generated by the prior encoder layers. The *Decoder Convolution* block consists of convolution layers.

The inputs to the Convolution blocks are provided in the form of a binary image consisting of a grid-based sequence of zeros and ones. Domain points contained within the geometry are labeled with 0’s while the remaining domain points are labeled with 1’s. The U-Net convolution modules encode the multi-spatial embedding of the object’s shape

and map the geometric shape represented as a binary image and the output hypersonic field around the object. We condition the model further by using the two parameters defining the geometry: the lengths of major ( $a$ ) and minor axes ( $b$ ) of the semi-ellipse. These parameters are used as inputs to the MLP block. The output embeddings generated from the MLP block are combined with the outputs from each *Encoder* block through multiple projection layers. The combination process provides the conditioning for creating a bijective mapping between input and output functions. The final layer converts the Decoder embeddings to the appropriate size for reconstructing the output fields over the domain.



**Figure A3:** Parameter conditioned U-Net architecture used for predicting density field across varying geometric shapes. The inputs consist of the geometry parameters (major and minor axes of ellipse) and the binary representation of the geometry.

## Fourier Neural Operator for uniform grid

Fourier Neural Operator (FNO) introduced by Li et al. [16] utilizes Fourier transformations of the input functions to learn the operator mapping between input and output

for solving PDEs in a given domain  $D \subset \mathbb{R}^d$ . For two infinite-dimensional bounded spaces  $\mathcal{A} = \mathcal{A}(D; \mathbb{R}^{d_{in}})$  and  $\mathcal{U} = (\mathcal{U}; \mathbb{R}^{d_{out}})$ , the operator mapping  $\mathcal{G}_\theta : \mathcal{A} \rightarrow \mathcal{U}$  represents the approximation of the operator  $u_i = \mathcal{G}_*(a_i)$ , where  $\{a_i, u_i\}, i = 1, \dots, N$  are the pairs of data observations. Here, the input function  $v(x) \in \mathcal{A}$  is first lifted to a higher dimension through the transformation  $\mathcal{P}(v(x)) : \mathbb{R}^{d_{in}} \rightarrow \mathbb{R}^{d_z}$  using a shallow fully-connected linear layer. A sequence of Fourier integral operations follows this applied iteratively on  $v_l(x) \in \mathbb{R}^{d_z}$ , defined as

$$\mathcal{K}(\phi)v_l(x) := \mathcal{F}^{-1}[\mathcal{R}_\phi \cdot (\mathcal{F}v_l)](x), \quad \forall x \in D \quad (\text{A2.9})$$

where  $\mathcal{R}_\phi$  represents the Fourier transformation of a periodic function  $\kappa : \bar{D} \rightarrow \mathbb{R}^{d_v \times d_v}$  with parameters  $\phi$ ,  $\mathcal{F}$  and  $\mathcal{F}^{-1}$  represent the Fourier and inverse Fourier transformations respectively. For frequency  $k$ ,  $(\mathcal{F}v_l)(k) \in \mathbb{C}^{d_v}$  and  $\mathcal{R}_\phi(k) \in \mathbb{C}^{d_v \times d_v}$  and since  $\kappa$  is assumed to be periodic, it admits Fourier series expansion which allows working with discrete modes  $k \in \mathbb{Z}^d$ . The Fourier transform  $\mathcal{F}$  is applied to each channel of  $v_l(x)$ , and the higher modes are truncated to filter the high-frequency elements and include only modes  $k$  within:

$$k_{max} = Z_{k_{max}} := \{k \in \mathbb{Z}^d : |k_j| \leq k_{max,i}, i = 1, \dots, d\} \quad (\text{A2.10})$$

For each mode of  $\mathcal{F}(v_l)$ , a different weight matrix  $\mathcal{R}_\phi \in \mathbb{C}^{d_v \times d_v}$  is applied to form a complex-valued weight tensor  $\mathcal{R}_l \in \mathbb{C}^{k \times d_v \times d_v}$ . This output of each Fourier layer is combined linearly with the residual projection of  $v_l(x) \in \mathbb{R}^{d_z \times d_z}$  through weight matrices  $W_l \in \mathbb{R}^{d_v \times d_v}$ . Thus, the output of the  $l^{th}$  Fourier layer is given by

$$v_{l+1} = \sigma(\mathcal{F}^{-1}(\mathcal{R}_l \cdot \mathcal{F}(v_l)) + W_l \cdot v_l + \mathbf{b}_l), \quad (\text{A2.11})$$

where  $\sigma$  is a non-linear activation and  $\mathbf{b}_l \in \mathbb{R}^{d_z}$  represents the layer bias.

In this work, the input to FNO is generated by concatenating the grayscale binary image  $(256 \times 256 \times 1)$  and the domain coordinates  $\{x, y\} \in \mathbb{R}^2$  to create a three-channelled

input. For FNO, we utilize the vanilla version of FNO implementation<sup>1</sup> and modify the network architecture and hyper-parameters to achieve lowest errors during validation. The discretized input ( $a_i$ ) and output function ( $u_i$ ) are normalized to a scale of 0 – 1 using the transformation  $x_{norm} = [(x - x_{min})/(x_{max} - x_{min})]$ . The framework consists of five spectral convolution layers that are initialized as complex weights  $W_{conv} \in \mathbb{C}^{d_{in} \times d_{out} \times k_1 \times k_2}$  multiplied with a scaling factor  $\frac{1}{d_{in} + d_{out}}$ . The input  $v(x)$  is projected into a higher dimension through initial transformation  $\mathcal{P}(v(x) : \mathbb{R}^3 \rightarrow \mathbb{R}^{32})$ . In the Fourier layer, this high dimension embedding  $v_l(x)$  is transformed using a Fourier transformation  $\mathcal{F}v_l(x)$  where the higher spectrum modes  $k_1$  and  $k_2$  are filtered before performing a dot product with  $W_{conv}$ . The Fourier space embedding is converted back into a real domain using an inverse Fourier transformation  $\mathcal{F}^{-1}$ . The output from the Fourier layers is combined with the residual projection of  $v_l(x)$  as per Equation A2.11. The output is obtained by passing the embeddings obtained from the Fourier layers through two fully connected layers. Notably, the activation function is required only outside the Fourier layers, and here, we use ReLU as our choice of activation. We use Adam as the choice of optimizer with an initial learning rate of  $1e^{-3}$  with a weight decay rate of  $1e^{-4}$ . The network is trained for 2500 epochs using relative  $\mathcal{L}_2$  norm as the loss function. Architecture details for the FNO framework are shown in Table A7.

**Table A7:** *Architecture details for FNO*

| Layer type                   | Number of layers | Filter size/Number of nodes | Filtering modes         | Activation |
|------------------------------|------------------|-----------------------------|-------------------------|------------|
| Spectral convolution (Wconv) | 5                | [64, 64, 128, 128, 256]     | [64, 64, 128, 128, 256] | None       |
| 2D convolution               | 5                | [64, 64, 128, 128, 256]     | None                    | None       |
| Fully connected              | 2                | [128, 4]                    | None                    | ReLU       |

## MeshGraphNets for unstructured irregular grid

MeshGraphNets combines graph neural networks (GNNs) with mesh-based geometric representations of simulation domains for capturing spatial relationships. Unlike conventional convolution-based frameworks, graph-based frameworks can operate directly on the

<sup>1</sup>[https://github.com/neuraloperator/Geo-FNO/blob/main/airfoils/naca\\_interp\\_fno.py](https://github.com/neuraloperator/Geo-FNO/blob/main/airfoils/naca_interp_fno.py)

mesh data that is commonly used for scientific tasks and, hence, do not necessarily require uniform grids. MeshGraphNets utilize the conventional messaging passing scheme in an Encoder-Process-Decoder framework [302] for mapping the input graph to an output graph with the same structure but different node features. Here, the graph nodes correspond to the mesh nodes, while the graph edges correspond to the edge connections between the mesh nodes. The encoder  $\mathcal{I} : \mathcal{X} \rightarrow \mathcal{G}$  embeds the node features  $X$  as latent graph,  $G^l = \mathcal{I}(X)$ , where  $G := \{V, E, X\}$  with  $V := v_1, v_2, \dots, v_N$  representing the spatial location of mesh points and  $E := \{e_{ij} : (i, j) \in V \times V\}$  representing the edges connecting the mesh points, and  $X \in \mathbb{R}^{N \times d}$  denoting the features of each node (if available). Separate encoders are used to encode the node and edge features from the graph. The edge attributes for each edge are determined by calculating the spatial separation between the node pair  $\{v_i, v_j\}$  connected by the edge and appending it with the norm  $\|v_i - v_j\|_{\mathcal{L}_2}$ . The encoder  $I$  embeds the node and edge feature into a latent vector at each node and edge using a sequence of MLPs. The processor consists of sequential, fully connected layers for message passing and node and edge features aggregation. The mesh edge  $e_{ij}$  and node embeddings  $v_i$  are updated by

$$\begin{aligned} e'_{ij} &\leftarrow \mathcal{N}_{\theta_E}(e_{ij}, v_i, v_j) \\ v'_i &\leftarrow \mathcal{N}_{\theta_V}(v_i, \sum_j e'_{ij}) + v_i \end{aligned} \tag{A2.12}$$

where  $\mathcal{N}_{\theta_E}$  and  $\mathcal{N}_{\theta_V}$  represent the processor MLP layers for edge and node, respectively. The decoder module uses a series of fully connected networks and decodes the latent node embeddings generated by the processor into one or more output node features  $p_i$ . The network is trained using a mean-squared error loss function.

In this work, we use the nodal coordinates of the mesh as node features  $v_i$  and the edge connectivity matrix as inputs to MeshGraphNet. The output labels consist of the four state variables  $\rho_i, u_i, v_i, p_i$  defined at each node. The node and edge encoding encoder consists of five sequential linear layers with ReLU6 activation function [303]. We

incorporate Dropouts after the first three encoder layers to prevent overfitting, followed by a layer normalization before applying the activation. The processor layer consists of eight sequential MLP layers with layer normalization and ReLU activation for processing the edge and node vectors. The decoder module contains four MLP layers, with layer normalization and ReLU6 activation applied to every layer except the output. Table A8 shows the network details used for this study. We choose the Adam optimizer with an initial learning rate of  $1e^{-3}$  with weight decay  $\lambda = 0.005$  and train the network for 5000 epochs with a batch size of fourteen.

**Table A8:** *Details of MeshGraphNet architecture.*

| Layer type | Number of layers | Number of nodes                  | Activation | Layernorm | Dropout |
|------------|------------------|----------------------------------|------------|-----------|---------|
| Encoder    | 5                | [32, 64, 64, 64, 64]             | ReLU6      | Yes       | Yes     |
| Processor  | 8                | [64, 64, 64, 64, 64, 64, 64, 64] | ReLU       | Yes       | None    |
| Decoder    | 4                | [64, 64, 64, 4]                  | ReLU6      | Yes       | None    |

## A3 Appenedix for Chapter 7

### A3.1 Knowledge Graph information retrieval

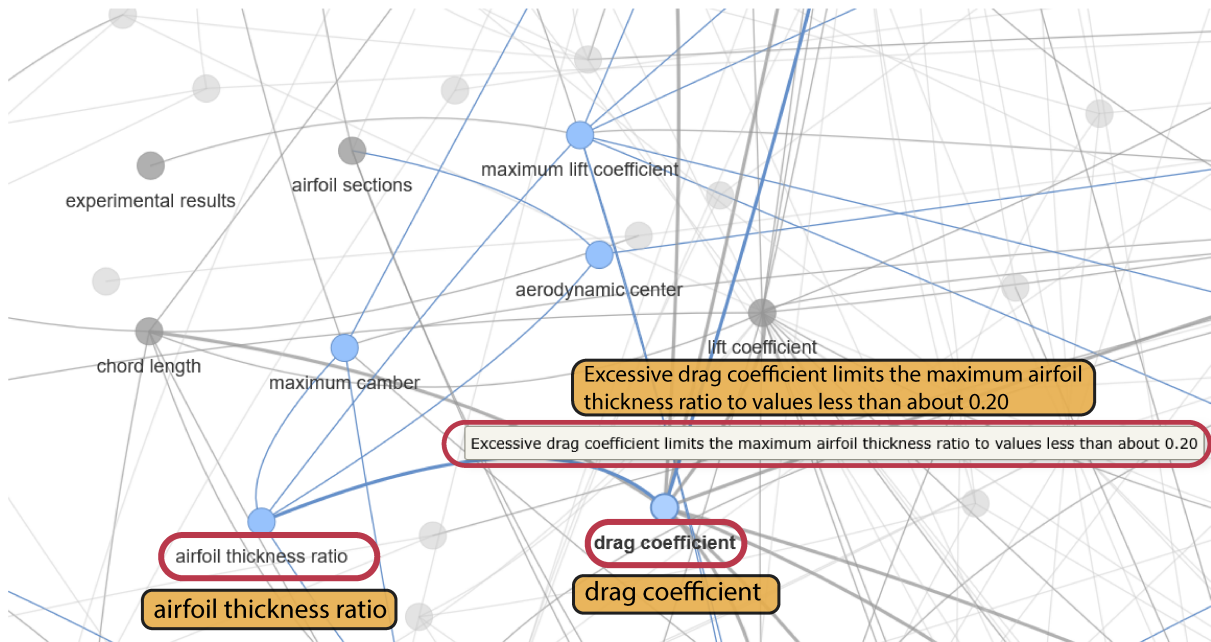
To validate and demonstrate the agent’s ability to effectively retrieve and apply knowledge, we implemented a mechanism to trace the provenance of its generated outputs. The System Engineer agent’s instructions were augmented to require explicit citation for any information retrieved from the knowledge graph via the Retrieval-Augmented Generation (RAG) framework. This modification compels the agent to label knowledge within its response using the format ‘(*src:KG, retrieved knowledge info*)’, providing a transparent link between its reasoning and the underlying knowledge base. We illustrate this capability through two examples from the technical requirements generation task.

First, we examine the derivation of a quantitative constraint for airfoil thickness. Responding to a general directive to use NACA 4-series designs, the agent retrieved a critical relationship from the knowledge graph connecting the nodes ‘airfoil thickness ratio’ and ‘drag coefficient’. The specific fact, "Excessive drag coefficient limits the maximum airfoil thickness ratio to values less than about 0.20," was directly utilized to establish an upper bound for the design parameter. As shown in the agent’s response, this led to the concrete requirement constraining thickness ‘t’ to  $\leq 20\%$ . This process, visually corroborated by the graph excerpt in Figure A4, showcases the agent’s capacity to ground design parameters in fundamental principles stored within the KG.

#### Knowledge Graph trace during requirement generation for thickness

\*\*\*Maximum Thickness (t)\*\*: The last two digits of the NACA 4-digit series, representing the maximum thickness as a percentage of the chord. The design space for ‘t’ shall be constrained to  $6\% \leq t \leq 20\%$  to manage drag at  $M=0.6$  and avoid excessive drag coefficients (*src:KG, airfoil thickness ratio, drag coefficient: Excessive drag coefficient limits the maximum airfoil thickness ratio to values less than about 0.20*. Reasoning: This provides a practical upper limit for the thickness ratio to avoid excessive drag, which is crucial for maximizing L/D)."

In a second example, the agent formulates a qualitative requirement for manufacturability. By querying the graph, the agent identified the conceptual link between ‘Design’ and ‘Manufacturability’ (Figure A5), retrieving the principle that manufacturability is a key design consideration. The agent further synthesized context from related graph



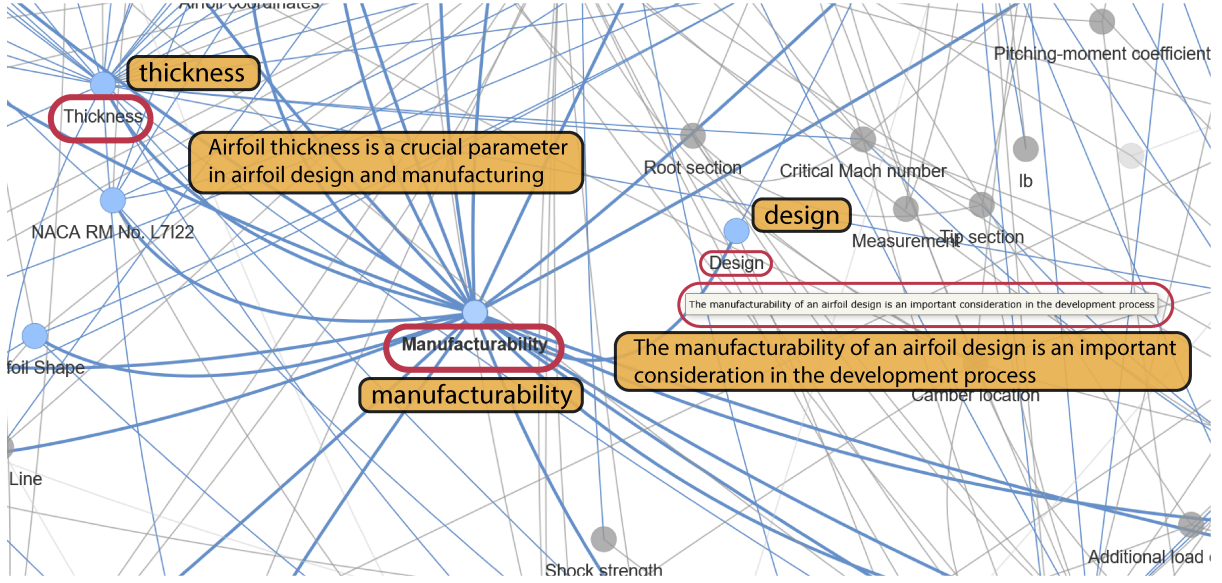
**Figure A4:** Section of the Systems Engineer's Knowledge Graph showing the two relevant nodes 'drag coefficient' and 'airfoil thickness ratio' retrieved to extract the required information using the ontological relationship between the two nodes for defining the maximum permissible airfoil thickness. The LLM prompt is used to extract semantically similar chunks from the graph data which is used as additional context for the final LLM response to the task.

segments, such as the connection between 'Manufacturability' and 'Thickness' (Figure A5), to reinforce its reasoning. This case highlights the system's ability to not only retrieve discrete facts but also to incorporate broader contextual connections and inter dependencies captured within the knowledge graph, ensuring a more grounded set of initial requirements.

#### Knowledge Graph trace during requirement generation for manufacturability

**Manufacturability:** The resulting airfoil geometry should be practical and feasible for manufacturing using conventional methods, avoiding features that would significantly increase production complexity or cost. (src:KG, Design, Manufacturability: The manufacturability of an airfoil design is an important consideration in the development process. Reasoning: A design, however aerodynamically efficient, is not practical if it cannot be manufactured. The thickness distribution is a key aspect of manufacturability)."





**Figure A5:** Section of the Systems Engineer’s Knowledge Graph showing the two relevant nodes ‘Design’ and ‘Manufacturability’ retrieved. The LLM prompt is used to extract semantically similar chunks from the graph data, from which the LLM module also extracts the connection between ‘Manufacturability’ and ‘Thickness’ of the airfoil.

The RAG process requires an embedder for creating the vector database for the graph data against which the agents prompts are compared for extracting semantically similar text chunks. In this work, we use OpenAI’s ‘text-embedding-3-small’ model for creating this vectorized database for both our Knowledge Graphs. The results we obtain for knowledge retrieval is based on this version of embedding model, and variations in knowledge retrieval may exist when using other LLM models for embedding or graph generation.

### **A3.2 Reasoning behind Systems Engineer agent’s design review process**

In this section, we provide an example showing the reasoning and steps used by the Systems Engineer agent while reviewing a design sample generated by the designer. The agent follows a structured reasoning workflow consisting of task interpretation, task decomposition, and action selection. The task assigned to the agent by the user is to use visual analysis along with geometric and aerodynamic data to evaluate the design’s suitability against technical requirements. The agent decomposes this task into the following action steps:

1. Image analysis of airfoil profile plot
2. Review of aerodynamic data provided by the Design Engineer
3. Review of geometric parameters provided by the Design Engineer
4. Evaluating design Quantities of Interest against Functional requirements
5. Evaluating design against Non-Functional requirements
6. Providing feedback and design improvement suggestions

The excerpt below shows details of the reasoning used during the review process.

## Excerpts from agent's reasoning during design review of sample ID-78

### Analyzing ID-78

**Final Result:** In the context of airfoil design, this could represent the upper and lower surfaces of an airfoil profile, with 'x' being the chordwise position and 'y' being the vertical coordinate.

**Thought:** The user wants me to analyze an airfoil profile (ID-78) based on an image, aerodynamic data, and geometric parameters. I need to assess its suitability against provided functional and non-functional requirements, provide feedback, and determine its validity.

Here's a breakdown of the steps:

- Analyze the image: The Vision Tool has described the image as a plot showing two curves, likely representing an airfoil profile. I need to infer the shape characteristics from this description.
- Review Aerodynamic Data: CD: 0.0084941151650085, CL: 1.142041730130238, CM: -0.3222857657483023
- Review Geometric Parameters: camber: 0.0762343152129009 (7.62%), camber location: 0.7696591844688178 (76.97% chord), thickness: 0.0459213124013521 (4.59%)
- Compare with Functional Requirements:
  - Airfoil Family: NACA 4-digit series geometry. (Need to check if the geometric parameters align with this, especially the thickness constraint).
  - Design Objective: Maximize Cl/Cd. Calculate Cl/Cd.
  - Operating Conditions:  $M=0.6$ ,  $Re=5 \times 10^6$ ,  $AoA=0$  degrees. (Aerodynamic data is provided for these conditions).
  - Geometric Parameters (NACA 4-digit Series Design Space)
    - \* Max Camber (m):  $0\% \leq m \leq 9\%$ . Current: 7.62%. (Valid)
    - \* Position of Max Camber (p):  $0 \leq p \leq 9$  (0% to 90% chord). Current: 76.97% ( $p=7.69$ , which is 7 in NACA 4-digit). (Valid)
    - \* Max Thickness (t):  $6\% \leq t \leq 20\%$ . Current: 4.59%. (Invalid - too thin)
  - Performance Metrics: Cl, Cd, Cl/Cd. (Provided)

## Excerpts from agent's reasoning during design review of sample ID-78 (cont'd)

- Compare with Non-functional Requirements:
  - Critical Mach Number:  $M_{crit} > 0.6$ . (No direct data, but thin airfoils generally have higher critical Mach numbers, however, the thickness is \*too\* thin, which might lead to other issues).
  - Boundary Layer Characteristics: Consider transition/separation at  $Re = 5 \times 10^6$  to minimize profile drag and ensure stable flow. (No direct data, but a very thin airfoil might have issues with boundary layer stability or structural integrity).
  - Pressure Distribution\*\*: Favorable pressure distribution for high lift/low drag at  $AoA=0$ ,  $M=0.6$ . (No direct data, but the high  $Cl/Cd$  suggests a good pressure distribution).
  - Computational Analysis Compatibility\*\*: Assumed compatible.
  - Manufacturability: Very thin airfoils can be challenging to manufacture and maintain structural integrity.
  - Robustness\*\*: Minor deviations should not lead to drastic performance degradation. (No direct data)
- Check  $Cl$  value: Is  $Cl < 0.5$ ? If yes, invalid. Current  $Cl = 1.142$ . (Valid,  $Cl > 0.5$ ).
- Formulate Feedback: Combine image analysis, aerodynamic performance, and parameter values. Suggest improvements.
- Determine Validity: Based on all checks.

Let's start processing the information.

- Image Analysis (from Vision Tool): The image shows a very thin airfoil profile, consistent with the calculated thickness of 4.6%. The presence of two curves (blue and red) likely represents the upper and lower surfaces, and the overall shape appears to be cambered, aligning with the given camber value. The profile seems smooth, which is generally good for aerodynamic performance.
- Geometric Parameters Analysis:
  - Camber (m): 0.0762343152129009 -> 7.6% (within 0-9% range). This corresponds to the first digit '7' in a NACA 4-digit series.
  - Camber Location (p): 0.7696591844688178 -> 77% chord (within 0-90% range). This corresponds to the second digit '7' in a NACA 4-digit series.
  - Thickness (t): 0.0459213124013521 -> 4.6% (NOT within 6-20% range). This corresponds to the last two digits '04' or '05' in a NACA 4-digit series.

The thickness constraint ( $6\% \leq t \leq 20\%$ ) is violated. The current thickness is 4.59%, which is too thin. This is a critical violation of the functional requirements.

- Aerodynamic Performance Analysis:
  - $Cl$ : 1.142041730130238 (Greater than 0.5, so valid by this rule of thumb).
  - $Cd$ : 0.0084941151650085
  - $Cl/Cd = 1.142041730130238 / 0.0084941151650085 = 134.45$

This is a very high lift-to-drag ratio, which is excellent for the design objective of maximizing  $Cl/Cd$ . The high lift coefficient at  $AoA=0$  is expected due to the significant camber (7.6%). The very low drag coefficient is also notable, possibly aided by the thin profile, but this thinness is a problem for the requirements.

- Suitability Assessment against Functional Requirements:
  - Airfoil Family: Assumed NACA 4-digit based on parameters, but the thickness constraint is violated.
  - Design Objective: Maximize  $Cl/Cd$ .  $Cl/Cd = 134.45$  is excellent.
  - Operating Conditions: Data provided for  $M=0.6$ ,  $Re = 5 \times 10^6$ ,  $AoA=0$ .
  - Geometric Parameters: Camber (m=7.62%): \*\*Meets\*\* (0-9%), Camber Location (p=76.97%): \*\*Meets\*\* (0-90%), Thickness (t=4.59%): \*\*FAILS\*\* (Required 6-20%). This is a major issue.
  - Performance Metrics:  $Cl$ ,  $Cd$ ,  $Cl/Cd$  are quantified.

## Excerpts from agent's reasoning during design review of sample ID-78 (cont'd)

- Suitability Assessment against Non-Functional Requirements:
  - Aerodynamic Performance (Critical Mach Number): A very thin airfoil (like 4.6%) would generally have a higher critical Mach number, which is good for  $M=0.6$ . However, the requirement specifies a minimum thickness of 6% to manage drag at  $M=0.6$ , implying that \*too\* thin might also be problematic or not the intended design space.
  - Boundary Layer Characteristics: A very thin airfoil might have challenges with boundary layer stability or transition at  $Re = 5 \times 10^6$ , potentially leading to early separation if not carefully designed. The low drag coefficient suggests good boundary layer behavior, but the extreme thinness raises concerns.
  - Pressure Distribution: The high  $Cl/Cd$  suggests a favorable pressure distribution.
  - Computational Analysis Compatibility: Assumed compatible.
  - Manufacturability: A 4.6% thick airfoil can be very difficult to manufacture and maintain structural integrity, especially for larger structures or under significant loads. This is a significant concern.
  - Robustness\*\*: No direct data, but extremely thin airfoils can sometimes be less robust to off-design conditions.
- Design Parameters and their effect:
  - Camber (0.0762): A non-zero camber is crucial for generating lift at  $AoA=0$ , as specified in the functional requirements. The current camber of 7.6% is substantial and contributes significantly to the high lift coefficient ( $Cl=1.142$ ).
  - Camber Location (0.7697): A camber location far aft (77% chord) can influence the pitching moment and pressure distribution. For a NACA 4-digit, 'p' is usually between 0 and 9. A value of 7 (70% chord) is common for aft-loaded airfoils. The current 76.97% is quite far back, which might lead to a more negative pitching moment ( $CM = -0.322$ ) and potentially affect stability.
  - Thickness (0.0459): The maximum thickness ratio is a critical parameter.
  - Impact on Drag: The requirement states "Excessive drag coefficient limits the maximum airfoil thickness ratio to values less than about 0.20". While the current thickness is well below 0.20, it is also below the \*minimum\* required thickness of 6%. Very thin airfoils can have low profile drag, but they can also lead to structural issues and potentially adverse pressure gradients if not designed carefully.
  - Impact on Lift: Thickness also affects the maximum lift coefficient.
  - Impact on Critical Mach Number: Thinner airfoils generally have higher critical Mach numbers, which is beneficial for  $M=0.6$ .
  - Impact on Manufacturability: A thickness of 4.6% is very thin and poses significant challenges for manufacturing and structural integrity.
- Cl value check:  $Cl = 1.142041730130238$ , which is greater than 0.5. So, the design is \*not\* considered invalid based on this rule of thumb.
- Overall Validity: The design is \*\*Invalid\*\* primarily due to the violation of the minimum thickness requirement (4.59% vs. required 6-20%). While the aerodynamic performance ( $Cl/Cd$ ) is excellent, this geometric constraint is fundamental. The very thin profile also raises concerns about manufacturability and structural integrity, which are non-functional requirements.
- Suggestions for Improvement: The primary suggestion is to increase the maximum thickness to be within the specified range of 6% to 20%. A thickness closer to the lower end of the acceptable range (e.g., 6-8%) might still offer good aerodynamic performance at  $M=0.6$  while meeting the structural and manufacturability considerations.
  - Increase Thickness: Aim for a thickness ratio ( $t$ ) between 0.06 and 0.08. This will bring the design into compliance with the functional requirements and improve manufacturability and structural robustness.
  - Re-evaluate Camber Location: While 77% chord for max camber is within the design space, it results in a significant negative pitching moment. Exploring a slightly more forward camber location (e.g., 40-60% chord, corresponding to '4' or '5' in NACA 4-digit) might help in reducing the pitching moment while still generating sufficient lift at  $AoA=0$ . However, the primary objective is  $Cl/Cd$  maximization, and the current configuration achieves a very high  $Cl/Cd$ .
  - Monitor Aerodynamic Performance: After adjusting thickness, re-evaluate  $Cl$ ,  $Cd$ , and  $Cl/Cd$  to ensure the design objective is still met. A slight increase in thickness might increase drag, but it should be manageable within the specified Mach number.

Feedback: ...

Valid: False

## A4 Appendices for Chapter 8

### A4.1 Related works

In this section, we provide an overview of existing works in three distinct but convergent research domains: Set-Based Design (SBD) for managing design space exploration, Risk-Based Design (RBD) for handling uncertainty, and the emerging field of Multi-Agent Systems (MAS) used in the context of engineering design.

As an alternative to traditional point-based design approach, SBD involves reasoning about and manipulating sets of designs [155]. Since its introduction, SBD has been applied to various engineering problems. Canbaz et al. [304] developed a collaborative SBD framework applied to a cantilever beam problem, demonstrating how sets can be managed and communicated in a distributed design environment. Hannapel, Vlahopoulos, and Singer [305] have extensively explored the intersection between SBD and Multidisciplinary Design Optimization (MDO), first proposing principles for including SBD in MDO and later detailing a formal implementation, demonstrating how the two paradigms can be used synergistically. Riaz et al. [306] applied a set-based approach to passenger aircraft family design, showcasing its utility in managing the immense complexity and inter-dependencies inherent in such systems. Similarly, Small et al. [307] provided a detailed case study of SBD for a UAV, highlighting its benefits in early-stage conceptual design trade-offs. Specking et al. [308] integrated SBD with Model-Based Systems Engineering for design space exploration and applied it to a UAV case study. Wade et al. [309] utilized SBD for designing engineered resilient systems, while McKenney et al. [310] demonstrated its effectiveness in adapting to evolving design requirements, a key advantage in long-duration projects. To enhance the rigor of set reduction, Georgiades et al. [311] proposed ADOPT, an augmented framework that integrates formal optimization techniques directly into the SBD workflow.

While SBD provides a framework for exploration, designing modern engineering systems requires a formal methodology for handling uncertainty. Traditional deterministic design

optimizes for a single operating point, while reliability-based design optimization (RBDO) targets a specific probability of failure. Risk-Based Design (RBD) extends these concepts by incorporating the consequences of failure, defining risk as a function of both failure probability and its associated cost or severity. Beck and Gomes [312] provided a comparative analysis, showing that risk-based optimization yields different and often more robust designs than deterministic or purely reliability-based methods. A foundational element of modern RBD is the use of coherent risk measures, such as the Conditional Value-at-Risk (CVaR) [159], (also known as superquantile) which quantifies the expected loss in the tail of a distribution. Rockafellar and Royset [157] established the theoretical underpinnings for using such measures in engineering design, providing a mathematically sound basis for risk-averse optimization. Royset et al. [160] proposed a Risk-Adaptive Set-Based Design framework, applying it to the shaping of a hydrofoil using CVaR as the risk metric for selecting design candidates. Chaudhuri et al. [313] investigated the use of various risk metrics, including Probability of Failure, CVaR, and Buffered Probability of Failure, and later developed methods for certifiable risk-based optimization [314], which aims to provide high-confidence guarantees on performance. The challenge of incorporating different types of uncertainty aleatory (inherent randomness) and epistemic (lack of knowledge) has been addressed by researchers like Rumpfkeil [315], who developed robust design methods for mixed uncertainties. Li et al. [316] developed a CVaR-based approach for uncertain MDO and later extended it to handle hybrid uncertainties [317]. Application-focused studies, such as the multi-objective robust design of airfoils by Padovan et al. [318], demonstrate the practical value of these methods in achieving designs that are insensitive to variations in operating conditions.

A new research direction in engineering design automation has been catalyzed by the advent of Large Language Models (LLMs). As a step further, integrating these LLMs into a MAS framework has now been shown to solve complex tasks [319, 320, 152]. Several frameworks have been proposed to structure this collaboration for design tasks. Obieke et al. [258] introduced AICED, a general framework for AI collaboration in design, while

Ding et al.’s DesignGPT [141] focuses specifically on multi-agent collaboration dynamics. Zhang et al. [145] demonstrated that agentic workflows in their iDesignGPT system can significantly boost engineering design productivity by automating complex, multi-step tasks. Application of MAS is rapidly evolving, and application examples can be seen across different design stages [265, 264, 262, 144, 143]. An important aspect of practical implementation of MAS is the management of interaction between the agents and external software/tools that exist in practice. This integration with external scientific computing tools is a recurring theme, with frameworks like My-CrunchGPT by Kumar et al. [146] aiming to create LLM-assisted platforms for broader scientific machine learning, which is directly applicable to engineering analysis. In a recent work, Kumar et al. [149] demonstrate the use of knowledge-guided MAS system for design, evaluation, and design modification of NACA airfoils, with a human-in-loop approach to ensure human control over the final solution candidate.

This research landscape presents a research opportunity: to develop a hybrid intelligence framework that leverages the strength of each domain to create a holistic design workflow. Specifically, there is a need for a multi-agent system that can orchestrate a design process grounded in Set-Based Design principles while automating some redundant tasks during concept evaluation and modification. Such a system could use LLM agents equipped with the tools necessary for the design workflow to perform the necessary functions in SBD and documenting the rationale for design selection. This approach can eventually automate the design space exploration process while ensuring that decisions are based on risk-informed quantitative evidence.

## A4.2 Bayesian surrogates

Unlike traditional neural networks that learn a single optimal value for each weight, a BNN learns a posterior distribution over its weights and biases,  $p(\mathbf{w}|\mathcal{D})$ , conditioned on the training data  $\mathcal{D}$ . As this true posterior is analytically intractable for deep neural



networks, we employ variational inference (VI) to approximate it with a more manageable distribution,  $q(\mathbf{w}|\theta)$ , parameterized by variational parameters  $\theta$  [1]. We place a standard normal distribution as a non-informative prior over the weights,  $p(\mathbf{w}) = \mathcal{N}(0, \mathbf{I})$ , which acts as a form of regularization. The variational posterior is modeled using a mean-field approximation, where each weight has an independent Gaussian distribution,  $q(w_i|\theta_i) = \mathcal{N}(w_i|\mu_i, \sigma_i^2)$ , with  $\theta_i = \{\mu_i, \sigma_i\}$  being the learnable mean and standard deviation for that weight. The training objective is to find the optimal parameters  $\theta^*$  by maximizing the Evidence Lower Bound (ELBO), which is equivalent to minimizing the Kullback-Leibler (KL) divergence between the approximate and true posteriors:

$$\mathcal{L}(\theta) = \mathbb{E}_{q(\mathbf{w}|\theta)}[\log p(\mathcal{D}|\mathbf{w})] - \text{KL}[q(\mathbf{w}|\theta) \parallel p(\mathbf{w})] \quad (\text{A4.1})$$

The network architecture itself is a fully-connected feed-forward model utilizing three ‘DenseFlipout’ layers, using a Leaky ReLU activation function with a negative slope of 0.2. The output of the network consists of two neurons, which directly parameterize the predictive distribution of the target coefficient. A softplus activation function is applied to the second output neuron to ensure the predicted standard deviation is always positive. For a given input vector  $\mathbf{x}$ , the network outputs the mean  $\mu(\mathbf{x})$  and standard deviation  $\sigma(\mathbf{x})$  of a Gaussian distribution, thereby capturing both epistemic uncertainty (through the distribution over weights) and aleatoric uncertainty (inherent noise in the data). The final predictive distribution for a new data point  $\mathbf{x}^*$  is obtained by marginalizing over the posterior of the weights,  $p(y^*|\mathbf{x}^*, \mathcal{D}) \approx \int p(y^*|\mathbf{x}^*, \mathbf{w})q(\mathbf{w}|\theta^*)d\mathbf{w}$ . For prediction, epistemic uncertainty is quantified through Monte Carlo sampling, where 200 forward passes are performed for each input, each time with a new set of weights drawn from their learned posterior distributions. The final deterministic prediction is taken as the mean of these samples, while their variance serves as a measure of the model’s confidence.

## Training data

For training our Bayesian surrogate, we utilize the CFD data published earlier in [10]. These simulations solve the Reynolds-Averaged Navier-Stokes (RANS) equations, coupled with the Spalart-Allmaras one-equation turbulence model to ensure accurate aerodynamic predictions. The dataset encompasses a total of 597 unique airfoil designs, each defined by a distinct combination of Class Shape Transformation (CST) parameters and operating conditions, specifically Mach number and angle of attack, which are varied within the bounds specified in Table A9. For the purpose of model training and validation, this dataset is partitioned into a pre-defined training set of 497 samples and a test set of 100 samples. Each sample in the dataset contains the integral aerodynamic performance coefficients: drag ( $C_D$ ), lift ( $C_L$ ), and moment ( $C_M$ ), as well as the detailed surface pressure distribution, represented by the pressure coefficient ( $C_P$ ) at numerous points along the airfoil chord. The Bayesian surrogate model is constructed to establish a probabilistic mapping from the input space of design and operational parameters,  $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^d$ , to the output space of aerodynamic coefficients,  $\mathbf{y} \in \mathcal{Y} \subset \mathbb{R}$ . Here, the input vector  $\mathbf{x}$  comprises the CST parameters and operating conditions, while the output is one of the scalar quantities  $\mathbf{y} = \{C_D, C_L, C_M\}$ . For the specific design space explored in this work, this purpose-built surrogate offers a higher level of fidelity and localized accuracy compared to more generalized, pre-trained models such as Neuralfoil.

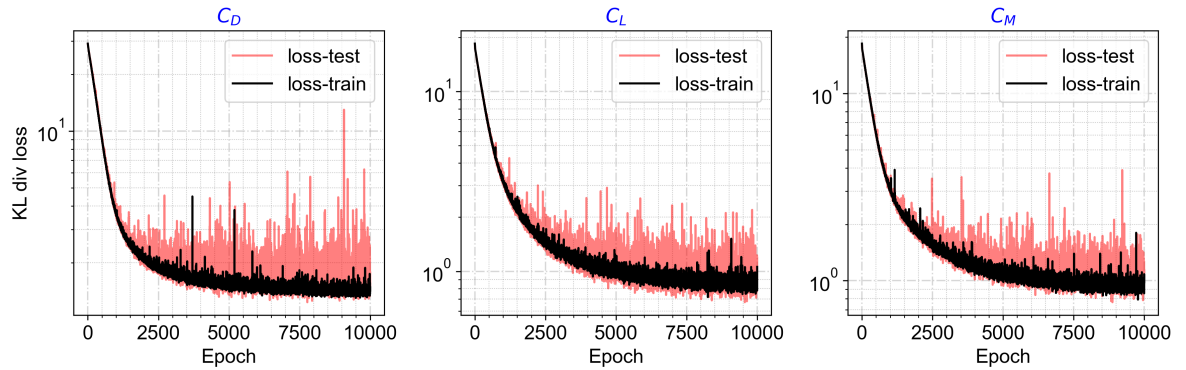
## Training results

The loss convergence plot for the three Bayesian networks trained separately is shown in figure A6. All models converge to a KL divergence loss value of  $\approx 1$  over 10,000 epochs. A prediction distribution for each test case is obtained by sampling the Bayesian models 200 times, resulting in a distribution of coefficient predictions for each airfoil design and operating conditions. In figure A7 - A9, we show the ensemble prediction results for the three coefficients across the test set. From this plot, we note that the Bayesian

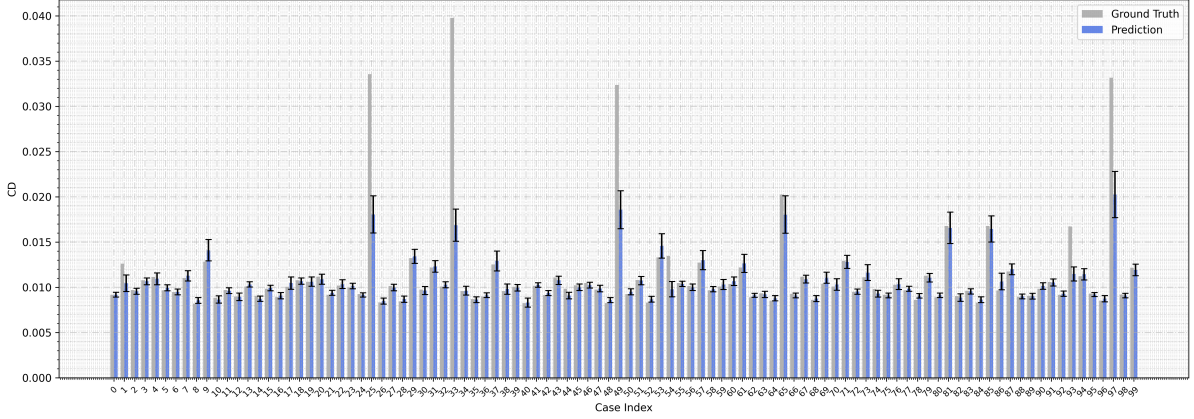
**Table A9:** Airfoil input parameter space showing the range of CST weights  $w$  and operating conditions used for our surrogate model training [10].

| Parameter | Lower bound     | Upper bound       |
|-----------|-----------------|-------------------|
| Ma        | 0.2             | 0.7               |
| AoA       | -3.0            | 5.0               |
| Re        | $1 \times 10^6$ | $6.5 \times 10^6$ |
| $w_{u,1}$ | 0.0644          | 0.1932            |
| $w_{u,2}$ | 0.0688          | 0.2064            |
| $w_{u,3}$ | 0.0961          | 0.2883            |
| $w_{u,4}$ | 0.0961          | 0.2882            |
| $w_{u,5}$ | 0.101           | 0.303             |
| $w_{l,2}$ | 0.068           | 0.2039            |
| $w_{l,3}$ | 0.1126          | 0.3377            |
| $w_{l,4}$ | 0.0381          | 0.1143            |
| $w_{l,5}$ | -0.0586         | -0.0195           |

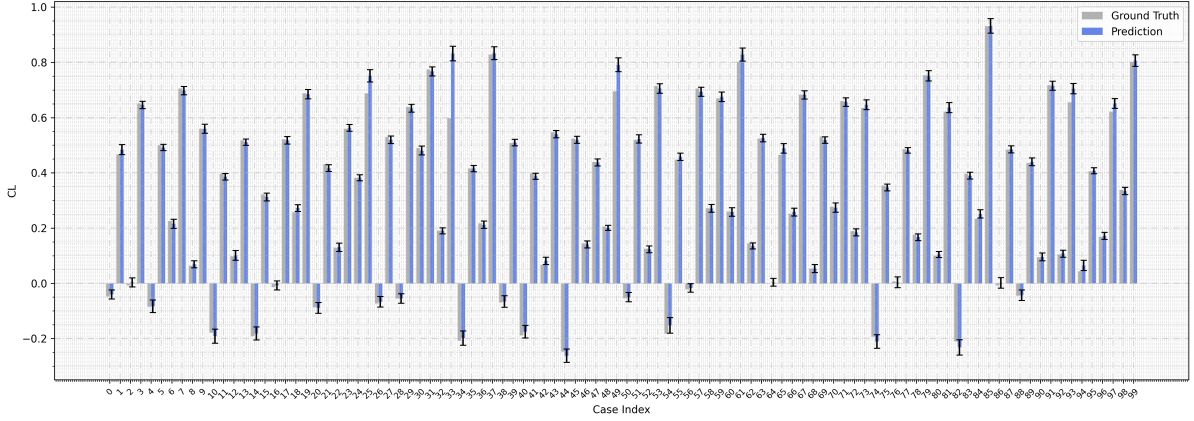
surrogates can accurately predict the aerodynamic coefficients for different airfoil designs and operating conditions. The  $C_D$  values for some samples, such as sample 31 shows significant departure from the average values that occur across different test samples. Some samples, in figure A8 also show negative lift, which occurs since the randomly generated airfoil designs may not always be aerodynamically consistent with expectations.



**Figure A6:** Loss evolution for the three Bayesian surrogates:  $C_D, C_L, C_M$ .



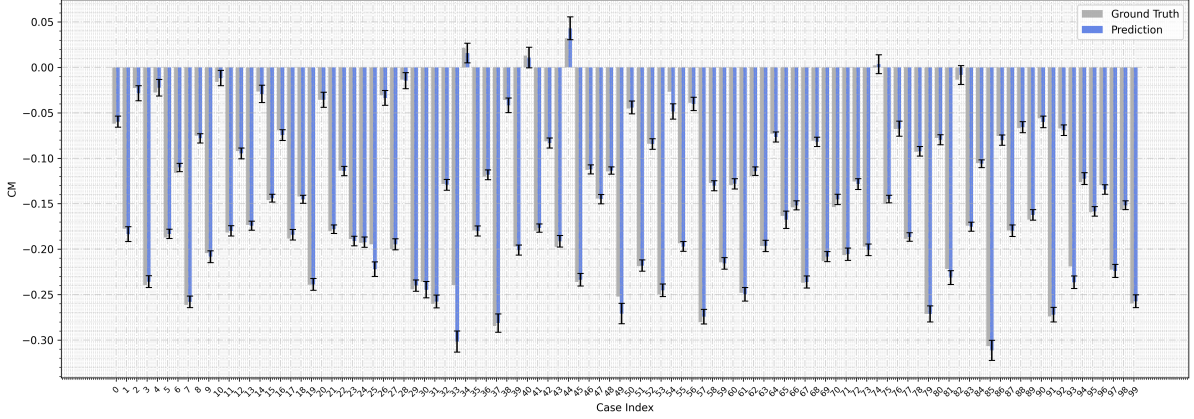
**Figure A7:** Ensemble prediction results from Bayesian surrogate for coefficient  $C_D$ . The black error bar shows the expected uncertainty in prediction combining both aleatoric and epistemic uncertainties.



**Figure A8:** Ensemble prediction results from Bayesian surrogate for coefficient  $C_L$ . The black error bar shows the expected uncertainty in prediction combining both aleatoric and epistemic uncertainties.

### A4.3 DeepONet surrogate

The DeepONet surrogate consists of two main components: a ‘branch’ network that encodes the input function parameters and a ‘trunk’ network that processes the spatial coordinates where the output function is evaluated. In our formulation, we define an operator  $\mathcal{G}$  that maps the combined geometry and flow conditions to the pressure coefficient function,  $C_P$ . The input to the operator is a vector  $\mathbf{u} \in \mathbb{R}^{12}$ , which concatenates the vector of Class Shape Transformation (CST) weights,  $\mathbf{w}$ , with the flow conditions: Mach number ( $Ma$ ), angle of attack ( $AoA$ ), and Reynolds number ( $Re$ ). The output of the operator,  $\mathcal{G}(\mathbf{u})$ , is a function that maps the airfoil surface coordinates,  $\mathbf{x} \in \mathbb{R}^2$ , to the

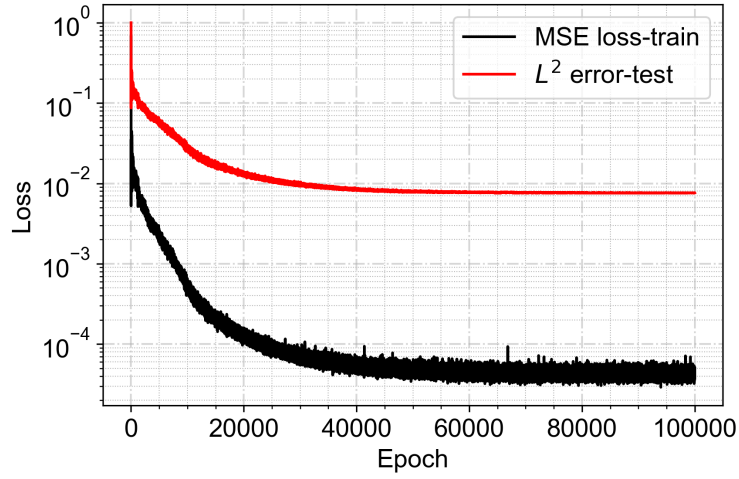


**Figure A9:** Ensemble prediction results from Bayesian surrogate for coefficient  $C_M$ . The black error bar shows the expected uncertainty in prediction combining both aleatoric and epistemic uncertainties.

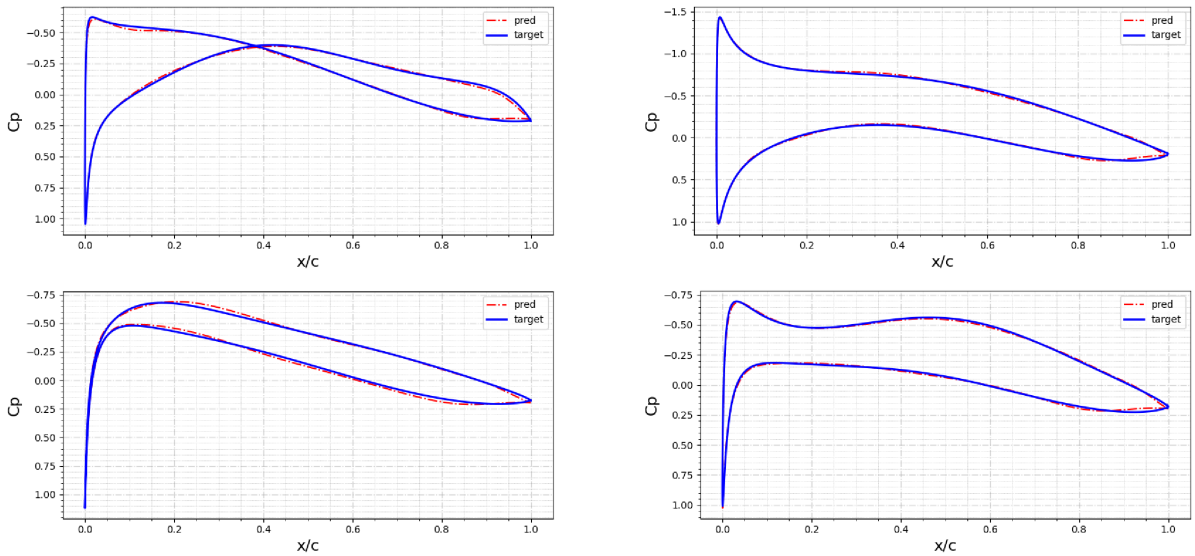
scalar pressure coefficient,  $C_P(\mathbf{x}, \mathbf{u}) \in \mathbb{R}$ . The DeepONet approximates this operator as  $\mathcal{G}(\mathbf{u})(\mathbf{x}) \approx \mathbf{b}(\mathbf{u})^T \mathbf{t}(\mathbf{x})$ , where  $\mathbf{b} : \mathbb{R}^{12} \rightarrow \mathbb{R}^p$  represents the branch network and  $\mathbf{t} : \mathbb{R}^2 \rightarrow \mathbb{R}^p$  is the trunk network, with both mapping to a shared latent space of dimension  $p$ . The training data, comprising surface coordinates and corresponding  $C_P$  values, was sourced from the same high-fidelity CFD dataset used for the Bayesian surrogates. Both the branch and trunk networks are implemented as Multi-Layer Perceptrons (MLPs) with five hidden layers configured with [64, 64, 128, 128, 128] units and utilize the Swish activation function throughout [321]. The latent dimension is set to  $p = 128$ . For preprocessing, branch inputs and target  $C_P$  values are normalized via min-max scaling, while trunk inputs are scaled to the range  $[-1, 1]$ . To enhance generalization, the branch network incorporates dropout with a rate of 0.05, while the trunk network employs Layer Normalization between successive layers. The model is trained by minimizing the Mean Squared Error (MSE) loss function using the Lion optimizer [322], with a batch size of 100. The learning rate is managed by an exponential decay schedule, starting at  $1 \times 10^{-3}$  and decaying with a rate of 0.96 every 500 steps. Model performance on unseen designs is assessed using the relative  $L^2$  error norm on the test set.

## Training results

Figure A10 shows the loss convergence for our DeepONet surrogate. The median  $L^2$  relative error, calculated individually across all 100 samples in the test set was found to be  $\approx 5.9\%$ . While the prediction was generally acceptable for most test cases, a few instances were observed where the prediction error was higher than the median error. However, the overall prediction from our DeepONet surrogate on this dataset was within acceptable threshold for this study. In figure A11, we show the prediction results for four test cases with our surrogate, where a reasonable fit to ground truth data is observed.



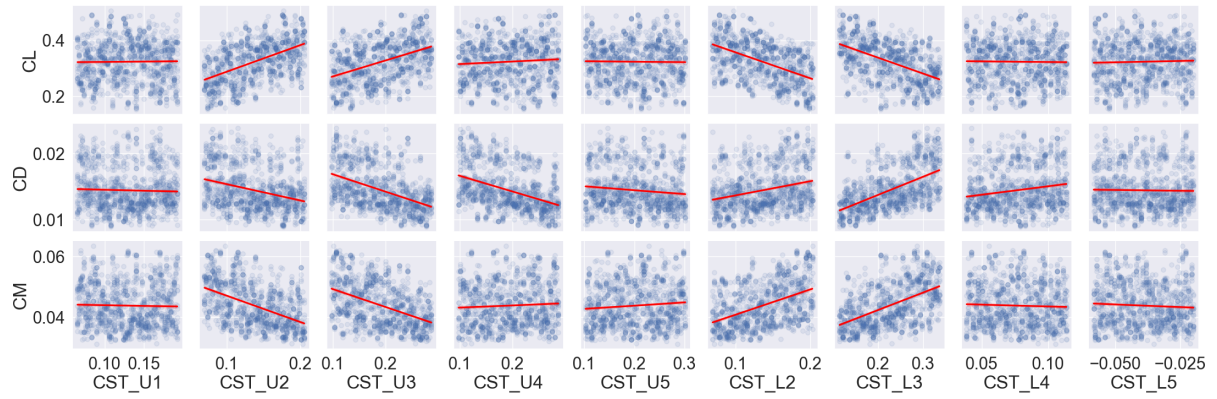
**Figure A10:** *Loss evolution for the DeepONet showing convergence.*



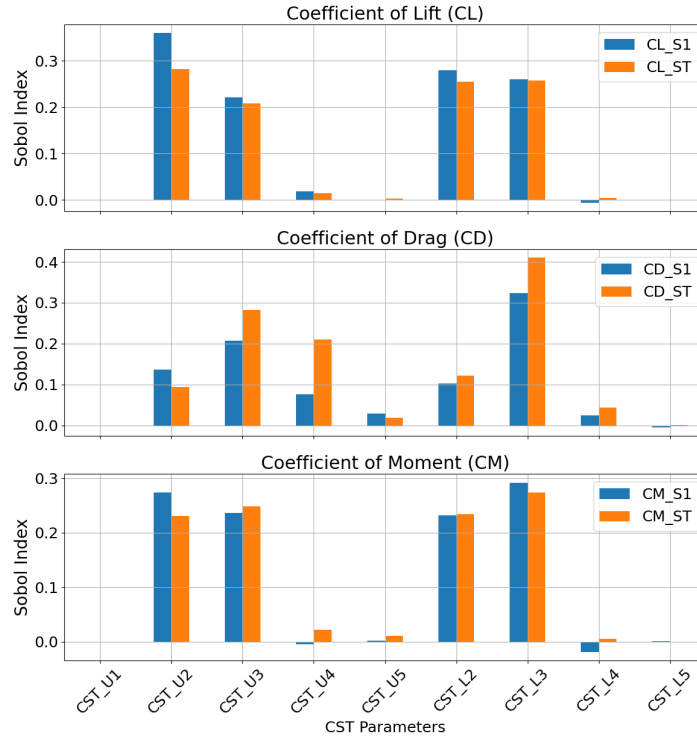
**Figure A11:** *Samples of DeepONet prediction for coefficient of pressure,  $C_P$ .*

#### A4.4 Stage 3: Sensitivity Analysis

In this section, we show the two plots that were generated during Stage 3: Sensitivity analysis of the 9 CST parameters and how they affect the aerodynamic performance coefficients. These results were generated by the Analyst agent, which re-uses these plots along with numerical values of the Sobol indices to determine the nature of response expected by increasing or decreasing a CST parameter.



**Figure A12:** Pairwise plot showing the individual effect of changing CST parameters on aerodynamic coefficients  $C_D$ ,  $C_L$ , and  $C_M$ .



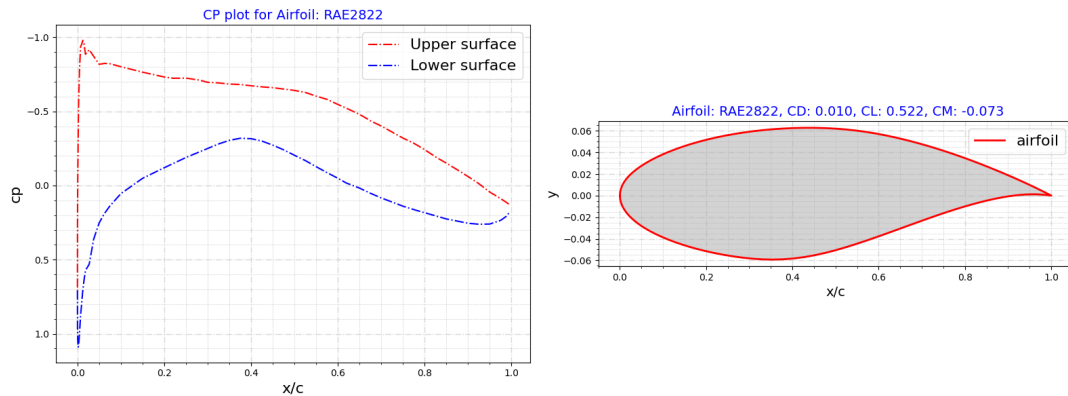
**Figure A13:** Plot showing magnitude of Sobol indices showing individual CST parameter effect ( $S_i$ ) and the total effect ( $S_{Ti}$ ) for each aerodynamic coefficient.



## A4.5 Stage 6: Automated Design review (additional information)

In this section, we provide additional information related to the automated design review phase performed by the Systems Engineer agent.

### RAE2822 benchmark



**Figure A14:** Plot showing the coefficient of pressure plot for benchmark airfoil RAE2822 at  $Re=6.3$  million,  $Ma=0.6$ , and  $AoA=2.57$  degrees [6]. The airfoil profile is shown alongside. The Systems Engineer agent compares the  $C_p$  plot along with utility score for the new airfoil designs against this benchmark and determines whether the design should be considered valid or not based on a set of heuristic rules defined by the human Manager.



## Engineering feedback generated during iterative design review and update loop

### Design feedback for Design ID-486 - iteration 3

#### “Pressure Distribution Curve Assessment:

1. The airfoil ID-486 exhibits a pronounced suction peak on the upper surface near the leading edge (around  $x/c = 0.05-0.1$ ), which contributes to a high lift coefficient but could potentially lead to early shock formation or flow separation at higher Mach numbers.
2. The pressure recovery on the upper surface is relatively smooth, indicating good flow behavior towards the trailing edge.
3. Compared to the benchmark RAE2822, ID-486 generates a higher pressure differential between the upper and lower surfaces, consistent with its higher CL. However, the benchmark's pressure distribution appears more tailored for supercritical flow with a more gradual suction peak.

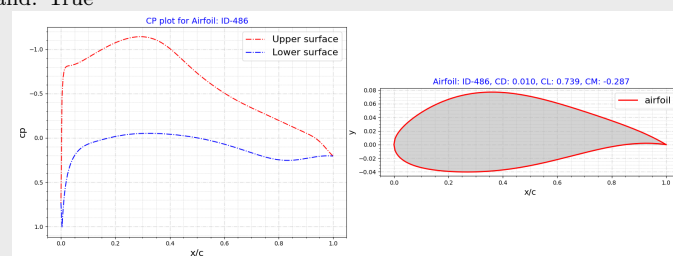
#### Airfoil Profile Assessment:

1. The airfoil ID-486 has a generally smooth and continuous profile, which is favorable for manufacturability and structural integrity.
2. The airfoil appears to have a reasonable thickness distribution, providing adequate volume for internal structures and maintaining structural integrity.
3. The overall shape is comparable to the benchmark RAE2822 in terms of non-functional requirements, with no apparent features that would hinder maintenance or service.

#### Design improvements:

1. Increase CST\_U2: This parameter has a highly beneficial effect, as increasing it will simultaneously increase the Coefficient of Lift (CL), decrease the Coefficient of Drag (CD), and increase the Coefficient of Moment (CM), making the pitching moment less negative and improving stability.
2. Decrease CST\_U3: Similar to CST\_U2, decreasing CST\_U3 will also lead to an increase in CL, a decrease in CD, and an increase in CM, contributing positively across all three aerodynamic coefficients.
3. Refine Leading Edge Curvature: To potentially mitigate the strong suction peak and further optimize drag, adjustments to the leading edge curvature, possibly through fine-tuning CST\_U1 or further adjustments to CST\_U2 and CST\_U3, could be explored to achieve a smoother pressure distribution while maintaining high lift.

Pressure rating: 4; Valid: True ”



**Figure A15:** Feedback for design ID-486 generated by the Systems Engineer with human Manager during third iteration after design modification.

## Design feedback for Design ID-486 - iteration 4

### “Pressure Distribution Curve:

1. Upper Surface Suction Peak: The airfoil ID-486 exhibits a strong suction peak near the leading edge ( $x/c$  0.05) on the upper surface, which is more pronounced than the benchmark RAE2822. This indicates higher local velocities and contributes significantly to the higher Coefficient of Lift (CL) observed for ID-486.
2. Pressure Recovery: Both the upper and lower surfaces show a smooth and gradual pressure recovery towards the trailing edge, suggesting that there is no significant flow separation, which is a positive indicator for aerodynamic efficiency.
3. Lift Generation: The larger pressure differential between the upper and lower surfaces, particularly due to the deeper suction on the upper surface, confirms the higher lift generation capability of ID-486 compared to the benchmark.

### Airfoil Profile:

1. Thickness and Camber: Airfoil ID-486 appears to be thicker and exhibits more camber, especially in the forward and mid-chord regions, compared to the benchmark RAE2822. This increased camber is consistent with its higher lift performance.
2. Structural Integrity: The thicker profile of ID-486 generally suggests better potential for structural integrity and allows for more internal volume for structural elements (e.g., spars) or systems, which is a favorable non-functional requirement.
3. Manufacturability: The overall smooth and continuous shape of the airfoil suggests good manufacturability, similar to the benchmark, without any sharp corners or abrupt changes in curvature that could pose manufacturing challenges.

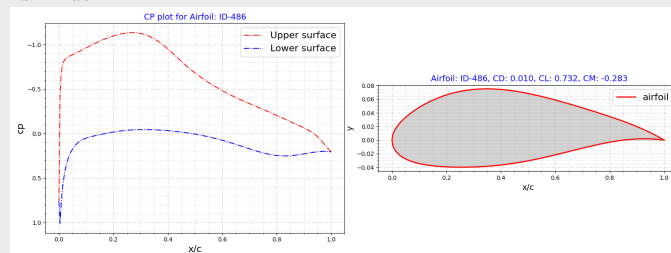
### Aerodynamic Performance and Utility Scores:

1. Coefficient of Lift (CL): ID-486 ( $CL=0.732$ ) significantly outperforms the benchmark ( $CL=0.522$ ), resulting in a much higher utility score for CL (ID-486: 0.5760 vs. Benchmark: 0.177).
2. Coefficient of Drag (CD): ID-486 ( $CD=0.0102$ ) has slightly higher drag than the benchmark ( $CD=0.010$ ). Consequently, its utility score for CD (ID-486: 0.5147) is marginally lower than the benchmark (0.518).
3. Coefficient of Moment (CM): ID-486 ( $CM=-0.2827$ ) exhibits a significantly more negative pitching moment compared to the benchmark ( $CM=-0.073$ ). This results in a much lower utility score for CM (ID-486: 0.0577 vs. Benchmark: 0.7566), indicating a stronger nose-down tendency which might require more trim effort.
4. Combined Utility Score: Despite the lower CM utility, ID-486 achieves a higher combined utility score (0.4539) than the benchmark (0.3955), primarily driven by its superior lift performance.

### Ways to Improve Performance through Design Modifications (*Focusing solely on CM as per manager's feedback*):

1. Increase CST\_L3: Based on the sensitivity analysis, increasing CST\_L3 is the most sensitive parameter for CM and shows a positive correlation, meaning it will increase CM (make it less negative).
2. Increase CST\_U2: This parameter is highly sensitive and indicates a positive correlation with CM. Increasing CST\_U2 will help increase CM.
3. Increase CST\_L2: This parameter is also highly sensitive and shows a positive correlation with CM. Increasing CST\_L2 will contribute to increasing CM.
4. Decrease CST\_U3: This parameter has a high sensitivity and a negative correlation with CM. Decreasing CST\_U3 will help increase CM.

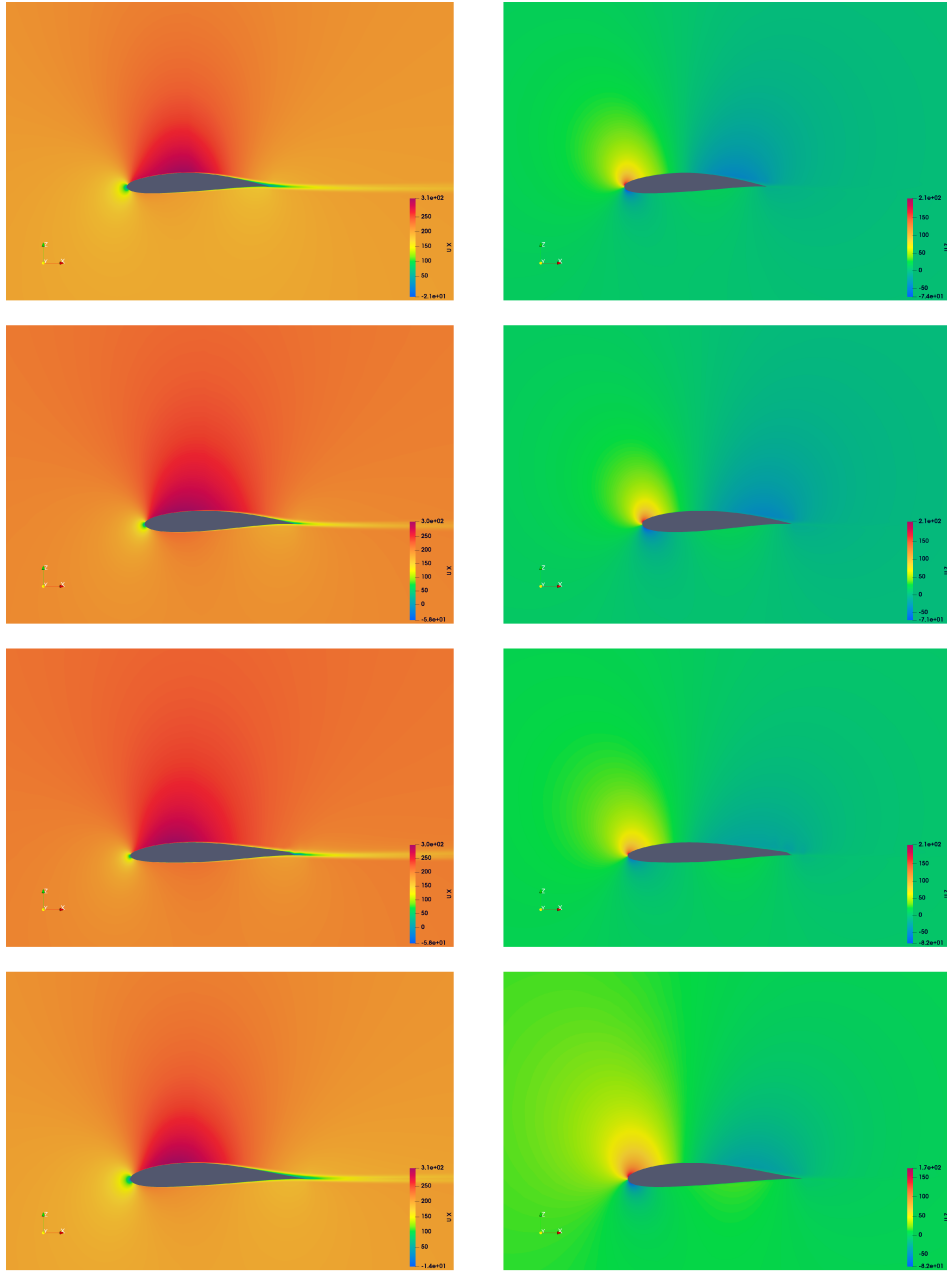
Pressure rating: 4; Valid: True ”



**Figure A16:** Feedback for design ID-486 generated by the Systems Engineer with human Manager during fourth iteration after design modification.

## A4.6 Stage 7: OpenFoam CFD additional results

In this section, we provide additional CFD simulation results for the final four design candidates selected at the end of the workflow. This CFD data is used, in conjunction with coefficient of pressure and performance metrics to determine the final airfoil design/s to be developed further.



**Figure A17:** Plot showing velocity fields for the final design candidates selected during the agentic workflow.

# Bibliography

- [1] S. Goswami, K. Kontolati, M. D. Shields, and G. E. Karniadakis, “Deep transfer operator learning for partial differential equations under conditional shift,” *Nature Machine Intelligence*, vol. 4, no. 12, pp. 1155–1164, 2022.
- [2] M. Raissi, P. Perdikaris, and G. Karniadakis, “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations,” *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.
- [3] S. Goswami, M. Yin, Y. Yu, and G. E. Karniadakis, “A physics-informed variational DeepONet for predicting crack path in quasi-brittle materials,” *Computer Methods in Applied Mechanics and Engineering*, vol. 391, p. 114587, 2022.
- [4] K. Shukla, V. Oommen, A. Peyvan, M. Penwarden, N. Plewacki, L. Bravo, A. Ghoshal, R. M. Kirby, and G. E. Karniadakis, “Deep neural operators as accurate surrogates for shape optimization,” *Engineering Applications of Artificial Intelligence*, vol. 129, p. 107615, 2024.
- [5] U. Ghia, K. Ghia, and C. Shin, “High-re solutions for incompressible flow using the navier-stokes equations and a multigrid method,” *Journal of Computational Physics*, vol. 48, pp. 387–411, 1982.

- [6] P. Cook, M. Firmin, and M. McDonald, *Aerofoil RAE 2822: Pressure Distributions, and Boundary Layer and Wake Measurements*. Experimental Data Base for Computer Program Assessment, AGARD Report ar 138, 1979.
- [7] Z. Zhang, Z. Zou, E. Kuhl, and G. E. Karniadakis, “Discovering a reaction–diffusion model for Alzheimer’s disease by combining PINNs with symbolic regression,” *Computer Methods in Applied Mechanics and Engineering*, vol. 419, p. 116647, 2024.
- [8] A. Peyvan, V. Oommen, A. D. Jagtap, and G. E. Karniadakis, “RiemannONets: Interpretable neural operators for Riemann problems,” *Computer Methods in Applied Mechanics and Engineering*, vol. 426, p. 116996, 2024.
- [9] B. Shih, A. Peyvan, Z. Zhang, and G. E. Karniadakis, “Transformers as neural operators for solutions of differential equations with finite regularity,” *Computer Methods in Applied Mechanics and Engineering*, vol. 434, p. 117560, 2025.
- [10] P. Bekemeyer, N. Hariharan, A. M. Wissink, and J. Cornelius, “Introduction of Applied Aerodynamics Surrogate Modeling Benchmark Cases,” in *AIAA SCITECH 2025 Forum*, p. 0036, 2025.
- [11] F. Tao, H. Zhang, and C. Zhang, “Advancements and challenges of digital twins in industry,” *Nature Computational Science*, vol. 4, no. 3, pp. 169–177, 2024.
- [12] R. Ghanem and P. D. Spanos, “Polynomial chaos in stochastic finite elements,” *Journal of Applied Mechanics*, vol. 57, pp. 197–202, 03 1990.
- [13] D. Xiu and G. E. Karniadakis, “The wiener–askey polynomial chaos for stochastic differential equations,” *SIAM Journal on Scientific Computing*, vol. 24, no. 2, pp. 619–644, 2002.
- [14] M. A. OLIVER and R. WEBSTER, “Kriging: a method of interpolation for geographical information systems,” *International journal of geographical information systems*, vol. 4, no. 3, pp. 313–332, 1990.

- [15] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, “Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators,” *Nature Machine Intelligence*, vol. 3, no. 3, pp. 218–229, 2021.
- [16] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, “Fourier neural operator for parametric partial differential equations,” *arXiv:2010.08895v3 [cs.LG]*, 2020.
- [17] N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, and A. Anandkumar, “Neural operator: Learning maps between function spaces with applications to pdes,” *Journal of Machine Learning Research*, vol. 24, no. 89, pp. 1–97, 2023.
- [18] K. Nath, V. Kumar, D. J. Smith, and G. E. Karniadakis, “A digital twin for diesel engines: Operator-infused physics-informed neural networks with transfer learning for engine health monitoring,” *Engineering Applications of Artificial Intelligence*, vol. 170, p. 114052, 2026.
- [19] Z. Li, D. Z. Huang, B. Liu, and A. Anandkumar, “Fourier neural operator with learned deformations for pdes on general geometries,” *Journal of Machine Learning Research*, vol. 24, no. 388, pp. 1–26, 2023.
- [20] V. Kumar, S. Goswami, D. Smith, and G. Karniadakis, “Real-time prediction of multiple output states in diesel engines using a deep neural operator framework,” *arXiv:2304.00567v2 [eess.SP]*, 2023.
- [21] V. Kumar, S. Goswami, K. Kontolati, M. D. Shields, and G. E. Karniadakis, “Synergistic Learning with Multi-Task DeepONet for Efficient PDE Problem Solving,” 2024.

- [22] A. Peyvan, V. Kumar, and G. E. Karniadakis, “Fusion-DeepONet: A Data-Efficient Neural Operator for Geometry-Dependent Hypersonic and Supersonic Flows,” *arXiv preprint arXiv:2501.01934*, 2025.
- [23] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. Battaglia, “Learning Mesh-Based Simulation with Graph Networks,” in *International Conference on Learning Representations*, 2021.
- [24] M. A. Nabian, C. Liu, R. Ranade, and S. Choudhry, “X-meshgraphnet: Scalable multi-scale graph neural networks for physics simulation,” *arXiv preprint arXiv:2411.17164*, 2024.
- [25] V. Kumar, J. Bi, C. N. Ngoc, V. Oancea, and G. E. Karniadakis, “Learning Nonlinear Responses in PET Bottle Buckling with a Hybrid DeepONet-Transolver Framework,” 2025.
- [26] H. Wu, H. Luo, H. Wang, J. Wang, and M. Long, “Transolver: a fast transformer solver for PDEs on general geometries,” in *Proceedings of the 41st International Conference on Machine Learning*, pp. 53681–53705, 2024.
- [27] L. Lu, X. Meng, S. Cai, Z. Mao, S. Goswami, Z. Zhang, and G. E. Karniadakis, “A comprehensive and fair comparison of two neural operators (with practical extensions) based on FAIR data,” *Computer Methods in Applied Mechanics and Engineering*, vol. 393, p. 114778, 2022.
- [28] S. Goswami, A. Bora, Y. Yu, and G. E. Karniadakis, “Physics-Informed Deep Neural Operator Networks,” in *Machine Learning in Modeling and Simulation: Methods and Applications*, pp. 219–254, Springer, 2023.
- [29] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, “Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators,” *Nature machine intelligence*, vol. 3, no. 3, pp. 218–229, 2021.

- [30] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, “Fourier Neural Operator for Parametric Partial Differential Equations,” *arXiv preprint arXiv:2010.08895*, 2020.
- [31] T. Tripura and S. Chakraborty, “Wavelet neural operator for solving parametric partial differential equations in computational mechanics problems,” *Computer Methods in Applied Mechanics and Engineering*, vol. 404, p. 115783, 2023.
- [32] Q. Cao, S. Goswami, and G. E. Karniadakis, “Laplace neural operator for solving differential equations,” *Nature Machine Intelligence*, pp. 1–10, 2024.
- [33] B. Raonic, R. Molinaro, T. Rohner, S. Mishra, and E. de Bezenac, “Convolutional Neural Operators,” in *ICLR 2023 Workshop on Physics for Machine Learning*, 2023.
- [34] R. Caruana, “Multitask Learning,” *Machine learning*, vol. 28, pp. 41–75, 1997.
- [35] P. Liu, X. Qiu, and X. Huang, “Deep multi-task learning with shared memory,” *arXiv preprint arXiv:1609.07222*, 2016.
- [36] M. Long, H. Zhu, J. Wang, and M. I. Jordan, “Deep transfer learning with joint adaptation networks,” in *International conference on machine learning*, pp. 2208–2217, PMLR, 2017.
- [37] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert, “Cross-stitch networks for multi-task learning,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3994–4003, 2016.
- [38] S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-Maron, M. Gimenez, Y. Sulsky, J. Kay, J. T. Springenberg, *et al.*, “A generalist agent,” *arXiv preprint arXiv:2205.06175*, 2022.
- [39] L. Yang, S. Liu, T. Meng, and S. J. Osher, “In-context operator learning with data prompts for differential equation problems,” *Proceedings of the National Academy of Sciences*, vol. 120, no. 39, p. e2310142120, 2023.



- [40] M. Herde, B. Raonić, T. Rohner, R. Käppeli, R. Molinaro, E. de Bézenac, and S. Mishra, “POSEIDON: Efficient Foundation Models for PDEs,” *arXiv preprint arXiv:2405.19101*, 2024.
- [41] L. Serrano, L. Le Boudec, A. Kassai Koupai, T. X. Wang, Y. Yin, J.-N. Vittaut, and P. Gallinari, “Operator learning with neural fields: Tackling pdes on general geometries,” in *Advances in Neural Information Processing Systems* (A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, eds.), vol. 36, pp. 70581–70611, Curran Associates, Inc., 2023.
- [42] T. Chen and H. Chen, “Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems,” *IEEE Transactions on Neural Networks*, vol. 6, no. 4, pp. 911–917, 1995.
- [43] P. C. Di Leoni, L. Lu, C. Meneveau, G. Karniadakis, and T. A. Zaki, “DeepONet prediction of linear instability waves in high-speed boundary layers,” *arXiv preprint arXiv:2105.08697*, 2021.
- [44] K. Kontolati, S. Goswami, M. D. Shields, and G. E. Karniadakis, “On the influence of over-parameterization in manifold based surrogates and deep neural operators,” *Journal of Computational Physics*, p. 112008, 2023.
- [45] V. Oommen, K. Shukla, S. Goswami, R. Dingreville, and G. E. Karniadakis, “Learning two-phase microstructure evolution using neural operators and autoencoder architectures,” *npj Computational Materials*, vol. 8, no. 1, p. 190, 2022.
- [46] Q. Cao, S. Goswami, T. Tripura, S. Chakraborty, and G. E. Karniadakis, “Deep neural operators can predict the real-time response of floating offshore structures under irregular waves,” *Computers & Structures*, vol. 291, p. 107228, 2024.

- [47] M. L. Taccari, H. Wang, S. Goswami, M. De Florio, J. Nuttall, X. Chen, and P. K. Jimack, “Developing a cost-effective emulator for groundwater flow modeling using deep neural operators,” *Journal of Hydrology*, vol. 630, p. 130551, 2024.
- [48] N. Borrel-Jensen, S. Goswami, A. P. Engsig-Karup, G. E. Karniadakis, and C.-H. Jeong, “Sound propagation in realistic interactive 3d scenes with parameterized sources using deep neural operators,” *Proceedings of the National Academy of Sciences*, vol. 121, no. 2, p. e2312159120, 2024.
- [49] S. De, M. Hassanaly, M. Reynolds, R. N. King, and A. Doostan, “Bi-fidelity Modeling of Uncertain and Partially Unknown Systems using DeepONets,” *arXiv preprint arXiv:2204.00997*, 2022.
- [50] L. Lu, R. Pestourie, S. G. Johnson, and G. Romano, “Multifidelity deep neural operators for efficient learning of partial differential equations with application to fast inverse design of nanoscale heat transport,” *arXiv preprint arXiv:2204.06684*, 2022.
- [51] A. A. Howard, M. Perego, G. E. Karniadakis, and P. Stinis, “Multifidelity Deep Operator Networks,” *arXiv preprint arXiv:2204.09157*, 2022.
- [52] P. Jin, S. Meng, and L. Lu, “MIONet: Learning multiple-input operators via tensor product,” *SIAM Journal on Scientific Computing*, vol. 44, no. 6, pp. A3490–A3514, 2022.
- [53] S. Goswami, D. S. Li, B. V. Rego, M. Latorre, J. D. Humphrey, and G. E. Karniadakis, “Neural operator learning of heterogeneous mechanobiological insults contributing to aortic aneurysms,” *Journal of the Royal Society Interface*, vol. 19, no. 193, p. 20220410, 2022.
- [54] E. Zhang, A. Kahana, E. Turkel, R. Ranade, J. Pathak, and G. E. Karniadakis, “A Hybrid Iterative Numerical Transferable Solver (HINTS) for PDEs Based on

- Deep Operator Network and Relaxation Methods,” *arXiv preprint arXiv:2208.13273*, 2022.
- [55] A. Kahana, E. Zhang, S. Goswami, G. Karniadakis, R. Ranade, and J. Pathak, “On the geometry transferability of the hybrid iterative numerical solver for differential equations,” *Computational Mechanics*, vol. 72, no. 3, pp. 471–484, 2023.
- [56] B. Bahmani, S. Goswami, I. G. Kevrekidis, and M. D. Shields, “A Resolution Independent Neural Operator,” *arXiv preprint arXiv:2407.13010*, 2024.
- [57] S. Goswami, K. Kontolati, M. D. Shields, and G. E. Karniadakis, “Deep transfer operator learning for partial differential equations under conditional shift,” *Nature Machine Intelligence*, pp. 1–10, 2022.
- [58] S. Wang, H. Wang, and P. Perdikaris, “Learning the solution operator of parametric partial differential equations with physics-informed DeepONets,” *Science advances*, vol. 7, no. 40, p. eabi8605, 2021.
- [59] L. Mandl, S. Goswami, L. Lambers, and T. Ricken, “Separable DeepONet: Breaking the Curse of Dimensionality in Physics-Informed Machine Learning,” *arXiv preprint arXiv:2407.15887*, 2024.
- [60] K. Kontolati, S. Goswami, G. Em Karniadakis, and M. D. Shields, “Learning nonlinear operators in latent spaces for real-time predictions of complex dynamics in physical systems,” *Nature Communications*, vol. 15, no. 1, p. 5101, 2024.
- [61] R. A. Fisher, “The wave of advance of advantageous genes,” *Annals of eugenics*, vol. 7, no. 4, pp. 355–369, 1937.
- [62] V. M. Tikhomirov, *Selected Works of AN Kolmogorov: Volume I: Mathematics and Mechanics*, vol. 25. Springer Science & Business Media, 1991.

- [63] J. Patade and S. Bhalekar, “Approximate analytical solutions of newell-whitehead-segel equation using a new iterative method,” *World Journal of Modelling and Simulation*, vol. 11, no. 2, pp. 94–103, 2015.
- [64] Y. Zeldovich, “Flame propagation in a substance reacting at initial temperature,” *Combustion and Flame*, vol. 39, no. 3, pp. 219–224, 1980.
- [65] J. He, S. Koric, D. Abueidda, A. Najafi, and I. Jasiuk, “Geom-DeepONet: A point-cloud-based deep operator network for field predictions on 3D parameterized geometries,” *Computer Methods in Applied Mechanics and Engineering*, vol. 429, p. 117130, 2024.
- [66] Paul, “3D Finite Element Analysis with MATLAB.” <https://www.mathworks.com/matlabcentral/fileexchange/50482-3d-finite-element-analysis-with-matlab>, 2024. Online; accessed 3 July 2024.
- [67] S. Liu, E. Johns, and A. J. Davison, “End-to-end multi-task learning with attention,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 1871–1880, 2019.
- [68] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, “Pointnet++: Deep hierarchical feature learning on point sets in a metric space,” *Advances in neural information processing systems*, vol. 30, 2017.
- [69] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen, “PointCNN: Convolution on x-transformed points,” *Advances in neural information processing systems*, vol. 31, 2018.
- [70] Y. Yang, C. Feng, Y. Shen, and D. Tian, “FoldingNet: Point Cloud Auto-encoder via Deep Grid Deformation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 206–215, 2018.

- [71] S. Koziel and L. Leifsson, “Multi-level cfd-based airfoil shape optimization with automated low-fidelity model selection,” *Procedia Computer Science*, vol. 18, pp. 889–898, 2013.
- [72] H. Shah, S. Hosder, S. Koziel, Y. A. Tesfahunegn, and L. Leifsson, “Multi-fidelity robust aerodynamic design optimization under mixed uncertainty,” *Aerospace Science and Technology*, vol. 45, pp. 17–29, 2015.
- [73] J. Ren, L. T. Leifsson, S. Koziel, and Y. Tesfahunegn, “Multi-fidelity aerodynamic shape optimization using manifold mapping,” in *57th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, p. 0419, 2016.
- [74] R. Skinner, A. Doostan, E. Peters, J. Evans, and K. E. Jansen, “An evaluation of multi-fidelity modeling efficiency on a parametric study of naca airfoils,” in *35th AIAA Applied Aerodynamics Conference*, p. 3260, 2017.
- [75] R. W. Skinner, A. Doostan, E. L. Peters, J. A. Evans, and K. E. Jansen, “Reduced-basis multifidelity approach for efficient parametric study of naca airfoils,” *AIAA Journal*, vol. 57, no. 4, pp. 1481–1491, 2019.
- [76] L. Leifsson and S. Koziel, “Multi-fidelity design optimization of transonic airfoils using physics-based surrogate modeling and shape-preserving response prediction,” *Journal of Computational Science*, vol. 1, no. 2, pp. 98–106, 2010.
- [77] J. Kou and W. Zhang, “Multi-fidelity modeling framework for nonlinear unsteady aerodynamics of airfoils,” *Applied Mathematical Modelling*, vol. 76, pp. 832–855, 2019.
- [78] P. Liao, W. Song, P. Du, and H. Zhao, “Multi-fidelity convolutional neural network surrogate model for aerodynamic optimization based on transfer learning,” *Physics of Fluids*, vol. 33, no. 12, 2021.

- [79] S. A. Renganathan, R. Maulik, and J. Ahuja, “Enhanced data efficiency using deep neural networks and gaussian processes for aerodynamic design optimization,” *Aerospace Science and Technology*, vol. 111, p. 106522, 2021.
- [80] J. Tao and G. Sun, “Application of deep learning based multi-fidelity surrogate model to robust aerodynamic design optimization,” *Aerospace Science and Technology*, vol. 92, pp. 722–737, 2019.
- [81] P. Wu, W. Yuan, L. Ji, L. Zhou, Z. Zhou, W. Feng, and Y. Guo, “Missile aerodynamic shape optimization design using deep neural networks,” *Aerospace Science and Technology*, vol. 126, p. 107640, 2022.
- [82] X. Zhang, F. Xie, T. Ji, Z. Zhu, and Y. Zheng, “Multi-fidelity deep neural network surrogate model for aerodynamic shape optimization,” *Computer Methods in Applied Mechanics and Engineering*, vol. 373, p. 113485, 2021.
- [83] K. Harada, D. Rajaram, and D. N. Mavris, “Application of transfer learning for multi-fidelity regression using physics-informed neural network on an airfoil,” in *AIAA SCITECH 2022 Forum*, p. 0386, 2022.
- [84] S. Bhatnagar, Y. Afshar, S. Pan, K. Duraisamy, and S. Kaushik, “Prediction of aerodynamic flow fields using convolutional neural networks,” *Computational Mechanics*, vol. 64, pp. 525–545, 2019.
- [85] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, “Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators,” *Nature Machine Intelligence*, vol. 3, no. 3, pp. 218–229, 2021.
- [86] Q. Cao, S. Goswami, and G. E. Karniadakis, “Laplace neural operator for solving differential equations,” *Nature Machine Intelligence*, vol. 6, no. 6, pp. 631–640, 2024.

- [87] T. Tripura and S. Chakraborty, “Wavelet neural operator for solving parametric partial differential equations in computational mechanics problems,” *Computer Methods in Applied Mechanics and Engineering*, vol. 404, p. 115783, 2023.
- [88] Z. Li, D. Z. Huang, B. Liu, and A. Anandkumar, “Fourier neural operator with learned deformations for pdes on general geometries,” *Journal of Machine Learning Research*, vol. 24, no. 388, pp. 1–26, 2023.
- [89] Z. Li, N. Kovachki, C. Choy, B. Li, J. Kossaifi, S. Otta, M. A. Nabian, M. Stadler, C. Hundt, K. Azizzadenesheli, *et al.*, “Geometry-informed neural operator for large-scale 3d pdes,” *Advances in Neural Information Processing Systems*, vol. 36, 2024.
- [90] T. Zhao, W. Qian, J. Lin, H. Chen, H. Ao, G. Chen, and L. He, “Learning mappings from iced airfoils to aerodynamic coefficients using a deep operator network,” *Journal of Aerospace Engineering*, vol. 36, no. 5, p. 04023035, 2023.
- [91] A. Peyvan, K. Shukla, J. Chan, and G. Karniadakis, “High-order methods for hypersonic flows with strong shocks and real chemistry,” *Journal of Computational Physics*, vol. 490, p. 112310, 2023.
- [92] A. S. Way, A. Sescu, E. Luke, and I. D. Dettwiler, “Hypernetwork based surrogate modeling of hypersonic glide vehicle aerothermodynamics,” in *AIAA AVIATION FORUM AND ASCEND 2024*, p. 4107, 2024.
- [93] J. A. Rataczak, I. D. Boyd, and J. W. McMahon, “Surrogate models for hypersonic aerothermodynamics and aerodynamics using gaussian process regression,” in *AIAA SCITECH 2024 Forum*, p. 0461, 2024.
- [94] C. Scherding, G. Rigas, D. Sipp, P. J. Schmid, and T. Sayadi, “An adaptive learning strategy for surrogate modeling of high-dimensional functions-application to unsteady hypersonic flows in chemical nonequilibrium,” *Computer Physics Communications*, vol. 307, p. 109404, 2025.

- [95] M. Schouler, Y. Prévèreaud, and L. Mieussens, “Machine learning based reduced models for the aerothermodynamic and aerodynamic wall quantities in hypersonic rarefied conditions,” *Acta Astronautica*, vol. 204, pp. 83–106, 2023.
- [96] Z. Zhao, C. Liu, Y. Li, Z. Chen, and X. Liu, “Diffeomorphism neural operator for various domains and parameters of partial differential equations,” *Communications Physics*, vol. 8, no. 1, p. 15, 2025.
- [97] M. Yin, N. Charon, R. Brody, L. Lu, N. Trayanova, and M. Maggioni, “A scalable framework for learning the geometry-dependent solution operators of partial differential equations,” *Nature computational science*, vol. 4, no. 12, pp. 928–940, 2024.
- [98] F. Casenave, “Mmcp: a mesh morphing gaussian process-based machine learning method for regression of physical problems under non-parameterized geometrical variability.” NeurIPS 2024 ML4CFD Competition: Harnessing Machine Learning for Computational Fluid Dynamics in Airfoil Design, 2024.
- [99] F. Casenave, B. Staber, and X. Roynard, “Mmcp: a mesh morphing gaussian process-based machine learning method for regression of physical problems under nonparametrized geometrical variability,” *Advances in Neural Information Processing Systems*, vol. 36, pp. 43972–43999, 2023.
- [100] G. Catalani, “Multiscale aerodynamic resolution invariant operator.” NeurIPS 2024 ML4CFD Competition: Harnessing Machine Learning for Computational Fluid Dynamics in Airfoil Design, 2024.
- [101] J. Helwig, “A geometry-aware message passing neural network for modeling aerodynamics over airfoils.” NeurIPS 2024 ML4CFD Competition: Harnessing Machine Learning for Computational Fluid Dynamics in Airfoil Design, 2024.



- [102] L. Cao, T. O’Leary-Roseberry, and O. Ghattas, “Derivative-informed neural operator acceleration of geometric mcmc for infinite-dimensional bayesian inverse problems,” *Journal of Machine Learning Research*, vol. 26, no. 78, pp. 1–68, 2025.
- [103] Y. Qiu, N. Bridges, and P. Chen, “Derivative-enhanced deep operator network,” *Advances in Neural Information Processing Systems*, vol. 37, pp. 20945–20981, 2024.
- [104] M. Schlottke-Lakemper, G. J. Gassner, H. Ranocha, A. R. Winters, and J. Chan, “Trixi.jl: Adaptive high-order numerical simulations of hyperbolic PDEs in Julia.” <https://github.com/trixi-framework/Trixi.jl>, 09 2021.
- [105] H. Ranocha, M. Schlottke-Lakemper, A. R. Winters, E. Faulhaber, J. Chan, and G. Gassner, “Adaptive numerical simulations with Trixi.jl: A case study of Julia for scientific computing,” *Proceedings of the JuliaCon Conferences*, vol. 1, no. 1, p. 77, 2022.
- [106] M. Schlottke-Lakemper, A. R. Winters, H. Ranocha, and G. J. Gassner, “A purely hyperbolic discontinuous Galerkin approach for self-gravitating gas dynamics,” *Journal of Computational Physics*, vol. 442, p. 110467, 06 2021.
- [107] S. Hennemann, A. M. Rueda-Ramírez, F. J. Hindenlang, and G. J. Gassner, “A provably entropy stable subcell shock capturing approach for high order split form DG for the compressible Euler equations,” *Journal of Computational Physics*, vol. 426, p. 109935, 2021.
- [108] M. H. Carpenter and C. A. Kennedy, “A fourth-order 2n-storage Runge-Kutta scheme,” NASA TM 109112, NASA, June 1994.
- [109] N. Sahoo, “Experiments on a blunt cone model in a hypersonic shock tunnel,” 2007.
- [110] M. D. McKay, R. J. Beckman, and W. J. Conover, “A comparison of three methods for selecting values of input variables in the analysis of output from a computer code,” *Technometrics*, vol. 42, no. 1, pp. 55–61, 2000.

- [111] G. B. Jacobs, *Numerical simulation of two-phase turbulent compressible flows with a multidomain spectral method*. University of Illinois at Chicago, 2003.
- [112] S. Wang, J. H. Seidman, S. Sankaran, H. Wang, and G. J. P. Paris, “Cvit: Continuous vision transformer for operator learning,” *arXiv preprint arXiv:2405.13998*, vol. 3, 2024.
- [113] J. Seidman, G. Kissas, P. Perdikaris, and G. J. Pappas, “Nomad: Nonlinear manifold decoders for operator learning,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 5601–5613, 2022.
- [114] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, “Geometric Deep Learning: Going beyond Euclidean data,” *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017.
- [115] L. Regenwetter, A. H. Nobari, and F. Ahmed, “Deep Generative Models in Engineering Design: A Review,” *Journal of Mechanical Design*, vol. 144, p. 071704, 03 2022.
- [116] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, “Fourier neural operator for parametric partial differential equations,” *arXiv preprint arXiv:2010.08895*, 2020.
- [117] Z. Li, D. Z. Huang, B. Liu, and A. Anandkumar, “Fourier neural operator with learned deformations for pdes on general geometries,” *Journal of Machine Learning Research*, vol. 24, no. 388, pp. 1–26, 2023.
- [118] N. Ashton, C. Mockett, M. Fuchs, L. Fliessbach, H. Hetmann, T. Knacke, N. Schönewald, V. Skaperdas, G. Fotiadis, A. Walle, *et al.*, “DrivAerML: High-fidelity computational fluid dynamics dataset for road-car external aerodynamics,” *arXiv preprint arXiv:2408.11969*, 2024.

- [119] V. Kumar, S. Goswami, K. Kontolati, M. D. Shields, and G. E. Karniadakis, “Synergistic learning with multi-task DeepONet for efficient PDE problem solving,” *Neural Networks*, vol. 184, p. 107113, 2025.
- [120] J. He, S. Koric, D. Abueidda, A. Najafi, and I. Jasiuk, “Geom-DeepONet: A point-cloud-based deep operator network for field predictions on 3D parameterized geometries,” *Computer Methods in Applied Mechanics and Engineering*, vol. 429, p. 117130, 2024.
- [121] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, “Pointnet: Deep learning on point sets for 3D classification and segmentation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 652–660, 2017.
- [122] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, “Pointnet++: Deep hierarchical feature learning on point sets in a metric space,” *Advances in neural information processing systems*, vol. 30, 2017.
- [123] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen, “Pointcnn: Convolution on x-transformed points,” *Advances in neural information processing systems*, vol. 31, 2018.
- [124] X. Ma, C. Qin, H. You, H. Ran, and Y. Fu, “Rethinking network design and local geometry in point cloud: A simple residual mlp framework,” *arXiv preprint arXiv:2202.07123*, 2022.
- [125] J. Park and N. Kang, “Point-DeepONet: A Deep Operator Network Integrating PointNet for Nonlinear Analysis of Non-Parametric 3D Geometries and Load Conditions,” *arXiv preprint arXiv:2412.18362*, 2024.
- [126] Z. Li, N. Kovachki, C. Choy, B. Li, J. Kossaifi, S. Otta, M. A. Nabian, M. Stadler, C. Hundt, K. Azizzadenesheli, *et al.*, “Geometry-informed neural operator for large-

- scale 3D PDEs,” *Advances in Neural Information Processing Systems*, vol. 36, pp. 35836–35854, 2023.
- [127] R. Ranade, M. A. Nabian, K. Tangsali, A. Kamenev, O. Hennigh, R. Cherukuri, and S. Choudhry, “Domino: A decomposable multi-scale iterative neural operator for modeling large scale engineering simulations,” *arXiv preprint arXiv:2501.13350*, 2025.
- [128] W. Zhong and H. Meidani, “Physics-informed geometry-aware neural operator,” *Computer Methods in Applied Mechanics and Engineering*, vol. 434, p. 117540, 2025.
- [129] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.
- [130] Z. Hao, Z. Wang, H. Su, C. Ying, Y. Dong, S. Liu, Z. Cheng, J. Song, and J. Zhu, “Gnot: A general neural operator transformer for operator learning,” in *International Conference on Machine Learning*, pp. 12556–12569, PMLR, 2023.
- [131] H. Luo, H. Wu, H. Zhou, L. Xing, Y. Di, J. Wang, and M. Long, “Transolver++: An Accurate Neural Solver for PDEs on Million-Scale Geometries,” *arXiv preprint arXiv:2502.02414*, 2025.
- [132] M. Elrefaie, F. Morar, A. Dai, and F. Ahmed, “Drivaernet++: A large-scale multimodal car dataset with computational fluid dynamics simulations and deep learning benchmarks,” *Advances in Neural Information Processing Systems*, vol. 37, pp. 499–536, 2024.
- [133] T. Wang and C. Wang, “Latent neural operator for solving forward and inverse pde problems,” *Advances in Neural Information Processing Systems*, vol. 37, pp. 33085–33107, 2024.

- [134] Dassault Systèmes Corp, “SIMULIA User Assistance 2025, Adaptive DOE Technique,” 2025.
- [135] Dassault Systèmes Corp, “SIMULIA User Assistance 2025, Abaqus Analysis Guide,” 2025.
- [136] T. Chen and H. Chen, “Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems,” *IEEE transactions on neural networks*, vol. 6, no. 4, pp. 911–917, 1995.
- [137] B. Sullivan and A. Kaszynski, “PyVista: 3D plotting and mesh analysis through a streamlined interface for the Visualization Toolkit (VTK),” *Journal of Open Source Software*, vol. 4, p. 1450, May 2019.
- [138] D. Hendrycks and K. Gimpel, “Gaussian error linear units (GELUs),” *arXiv preprint arXiv:1606.08415*, 2016.
- [139] L. N. Smith and N. Topin, “Super-convergence: Very fast training of neural networks using large learning rates,” in *Artificial intelligence and machine learning for multi-domain operations applications*, vol. 11006, pp. 369–386, SPIE, 2019.
- [140] M. Polanyi, “The Tacit Dimension,” in *Knowledge in Organisations* (L. Prusak, ed.), ch. 7, pp. 135–146, Boston: Butterworth-Heinemann, 1 ed., 1997.
- [141] S. Ding, X. Chen, Y. Fang, W. Liu, Y. Qiu, and C. Chai, “Designgpt: Multi-agent collaboration in design,” in *2023 16th International Symposium on Computational Intelligence and Design (ISCID)*, pp. 204–208, IEEE, 2023.
- [142] F. Ocker, S. Menzel, A. Sadik, and T. Rios, “From idea to CAD: a language model-driven multi-agent system for collaborative design,” *arXiv preprint arXiv:2503.04417*, 2025.

- [143] C. Picard, K. M. Edwards, A. C. Doris, B. Man, G. Giannone, M. F. Alam, and F. Ahmed, "From concept to manufacturing: evaluating vision-language models for engineering design," *Artificial Intelligence Review*, vol. 58, no. 9, p. 288, 2025.
- [144] M. Elrefaie, J. Qian, R. Wu, Q. Chen, A. Dai, and F. Ahmed, "AI Agents in Engineering Design: A Multi-Agent Framework for Aesthetic and Aerodynamic Car Design," in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 89237, p. V03BT03A048, American Society of Mechanical Engineers, 2025.
- [145] Z. Zhang, S. Liu, Y. Shen, Y. Zhang, Z. Hou, X. Wang, and J. Luo, "iDesignGPT: large language model agentic workflows boost engineering design." <https://doi.org/10.21203/rs.3.rs-5670522/v1>, 2025. Preprint (Version 1) available at Research Square.
- [146] V. Kumar, L. Gleyzer, A. Kahana, K. Shukla, and G. E. Karniadakis, "Mycrunchgpt: A LLM Assisted Framework for Scientific Machine Learning," *Journal of Machine Learning for Modeling and Computing*, vol. 4, no. 4, 2023.
- [147] M. Wooldridge, *An introduction to multiagent systems*. John wiley & sons, 2009.
- [148] D. Kinny and M. Georgeff, "Modelling and design of multi-agent systems," in *Intelligent Agents III Agent Theories, Architectures, and Languages*, pp. 1–20, Springer, 1997.
- [149] V. Kumar and G. E. Karniadakis, "Toward Autonomous Engineering Design: A Knowledge-Guided Multi-Agent Framework," *arXiv preprint arXiv:2511.03179*, 2025.
- [150] X. F. Zha, "A knowledge intensive multi-agent framework for cooperative/collaborative design modeling and decision support of assemblies," *Knowledge-Based Systems*, vol. 15, no. 8, pp. 493–506, 2002.

- [151] H. He, A. Balakrishnan, M. Eric, and P. Liang, “Learning symmetric collaborative dialogue agents with dynamic knowledge graph embeddings,” in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1766–1776, 2017.
- [152] A. Ghafarollahi and M. J. Buehler, “SciAgents: Automating Scientific Discovery Through Bioinspired Multi-Agent Intelligent Graph Reasoning,” *Advanced Materials*, vol. 37, no. 22, p. 2413523, 2025.
- [153] M. J. Buehler, “Accelerating scientific discovery with generative knowledge extraction, graph-based representation, and multimodal intelligent graph reasoning,” *Machine Learning: Science and Technology*, vol. 5, no. 3, p. 035083, 2024.
- [154] A. Hogan, E. Blomqvist, M. Cochez, C. d’Amato, G. D. Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier, *et al.*, “Knowledge Graphs,” *ACM Computing Surveys (Csur)*, vol. 54, no. 4, pp. 1–37, 2021.
- [155] A. Ward, J. K. Liker, J. J. Cristiano, and D. K. Sobek II, “The Second Toyota Paradox: How Delaying Decisions Can Make Better Cars Faster,” *Sloan Management Review*, 1995.
- [156] D. J. SINGER, N. DOERRY, and M. E. BUCKLEY, “What is set-based design?,” *Naval Engineers Journal*, vol. 121, no. 4, pp. 31–43, 2009.
- [157] R. T. Rockafellar and J. O. Royset, “Risk measures in engineering design under uncertainty,” in *International Conference on Applications of Statistics and Probability in Civil Engineering (ICASP)*, Jul 2015.
- [158] A. Saltelli, M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana, and S. Tarantola, *Variance-Based Methods*, ch. 4, pp. 155–182. John Wiley & Sons, Ltd, 2007.

- [159] R. T. Rockafellar and S. Uryasev, “Optimization of conditional value-at-risk,” *Journal of Risk*, vol. 2, pp. 21–42, 2000.
- [160] J. O. Royset, L. Bonfiglio, G. Vernengo, and S. Brizzolara, “Risk-adaptive set-based design and applications to shaping a hydrofoil,” *Journal of Mechanical Design*, vol. 139, p. 101403, 08 2017.
- [161] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.
- [162] J. Devlin, M. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” *arXiv:1810.04805v2 [cs.CL]*, 2018.
- [163] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. Liu, “Exploring the limits of transfer learning with a unified text-to-text transformer,” *The Journal of Machine Learning Research*, vol. 21, no. 1, pp. 5485–5551, 2020.
- [164] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, “Roberta: A robustly optimized bert pretraining approach,” *arXiv preprint arXiv:1907.11692*, 2019.
- [165] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, “Xlnet: Generalized autoregressive pretraining for language understanding,” *Advances in Neural Information Processing Systems*, vol. 32, 2019.
- [166] V. Kumar, A. Choudhary, and E. Cho, “Data augmentation using pre-trained transformer models,” *arXiv:2003.02245v2 [cs.CL]*, 2020.



- [167] S. Shah, S. Taj, Q. Ho, Y.-Y. Ou, *et al.*, “Gt-finder: Classify the family of glucose transporters with pre-trained BERT language models,” *Computers in Biology and Medicine*, vol. 131, p. 104259, 2021.
- [168] S. Rothe, S. Narayan, and A. Severyn, “Leveraging pre-trained checkpoints for sequence generation tasks,” *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 264–280, 2020.
- [169] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, “Language models are few-shot learners,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877–1901, 2020.
- [170] A. Blanco-Gonzalez, A. Cabezon, A. Seco-Gonzalez, D. Conde-Torres, P. Antelo-Riveiro, A. Pineiro, and R. Garcia-Fandino, “The role of ai in drug discovery: Challenges, opportunities, and strategies,” *arXiv:2212.08104v1 [cs.CL]*, 2022.
- [171] A. Bran, S. Cox, A. White, and P. Schwaller, “ChemCrow: Augmenting large-language models with chemistry tools,” *arXiv:2304.05376v4 [physics.chem-ph]*, 2023.
- [172] S. Frieder, L. Pinchetti, R. Griffiths, T. Salvatori, T. Lukasiewicz, P. C. Petersen, A. Chevalier, and J. Berner, “Mathematical capabilities of ChatGPT,” *arXiv:2301.13867v2 [cs.LG]*, 2023.
- [173] C. Leiter, R. Zhang, Y. Chen, J. Belouadi, D. Larionov, V. Fresen, and S. Eger, “ChatGPT: A meta-analysis after 2.5 months,” *arXiv:2302.13795v1 [cs.CL]*, 2023.
- [174] P. Maddigan and T. Susnjak, “Chat2vis: Generating data visualisations via natural language using ChatGPT, codex and GPT-3 large language models,” *arXiv:2302.02094v2 [cs.HC]*, 2023.
- [175] S. A. Prieto, E. Mengiste, and B. García de Soto, “Investigating the use of ChatGPT for the scheduling of construction projects,” *Buildings*, vol. 13, no. 4, p. 857, 2023.

- [176] H. Hassani and E. Silva, “The role of ChatGPT in data science: how ai-assisted conversational interfaces are revolutionizing the field,” *Big data and Cognitive Computing*, vol. 7, no. 2, p. 62, 2023.
- [177] L. Bishop, “A computer wrote this paper: what ChatGPT means for education, research, and writing,” 2023.
- [178] B. Buchanan and E. Feigenbaum, “Dendral and meta-dendral: Their applications dimension,” *Artificial intelligence*, vol. 11, no. 1-2, pp. 5–24, 1978.
- [179] M. Raissi, P. Perdikaris, and G. Karniadakis, “Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations,” *arXiv:1711.10561v1 [cs.AI]*, 2017.
- [180] F. Buonomici, M. Carfagni, R. Furferi, Y. Volpe, and L. Governi, “Generative design: an explorative study,” *Computer-Aided Design and Applications*, vol. 18, no. 1, pp. 144–155, 2020.
- [181] N. Kallioras and N. Lagaros, “Dzain: Deep learning based generative design,” *Procedia Manufacturing*, vol. 44, pp. 591–598, 2020.
- [182] S. Krish, “A practical generative design method,” *Computer-Aided Design*, vol. 43, no. 1, pp. 88–100, 2011.
- [183] T. Gruber and D. Russell, “Generative design rationale: Beyond the record and replay paradigm,” in *Design Rationale*, pp. 323–349, CRC Press, 2020.
- [184] K. Shea, R. Aish, and M. Gourtovaia, “Towards integrated performance-driven generative design tools,” *Automation in Construction*, vol. 14, no. 2, pp. 253–264, 2005.
- [185] W. Noble, “What is a support vector machine?,” *Nature Biotechnology*, vol. 24, no. 12, pp. 1565–1567, 2006.

- [186] V. Rodriguez-Galiano, M. Sanchez-Castillo, M. Chica-Olmo, and M. Chica-Rivas, “Machine learning predictive models for mineral prospectivity: An evaluation of neural networks, random forest, regression trees and support vector machines,” *Ore Geology Reviews*, vol. 71, pp. 804–818, 2015.
- [187] B. de Silva, K. Champion, M. Quade, J.-C. Loiseau, J. N. Kutz, and S. L. Brunton, “Pysindy: a python package for the sparse identification of nonlinear dynamics from data,” *arXiv:2004.08424v1 [math.DS]*, 2020.
- [188] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
- [189] NVIDIA, “NVIDIA Modulus: a Neural Network Framework,” 2023.
- [190] V. Kag, K. Seshasayanan, and V. Gopinath, “Physics-informed data based neural networks for two-dimensional turbulence,” *Physics of Fluids*, vol. 34, no. 5, p. 055130, 2022.
- [191] L. Yang, X. Meng, and G. Karniadakis, “B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data,” *Journal of Computational Physics*, vol. 425, p. 109913, 2021.
- [192] G. Pang, L. Lu, and G. Karniadakis, “fpinns: Fractional physics-informed neural networks,” *SIAM Journal on Scientific Computing*, vol. 41, no. 4, pp. A2603–A2626, 2019.
- [193] X. Meng, Z. Li, D. Zhang, and G. Karniadakis, “PPINN: Parareal physics-informed neural network for time-dependent PDEs,” *Computer Methods in Applied Mechanics and Engineering*, vol. 370, p. 113250, 2020.
- [194] E. Kharazmi, Z. Zhang, and G. Karniadakis, “hp-vpinns: Variational physics-informed neural networks with domain decomposition,” *Computer Methods in Applied Mechanics and Engineering*, vol. 374, p. 113547, 2021.

- [195] L. Yang, D. Zhang, and G. Karniadakis, “Physics-informed generative adversarial networks for stochastic differential equations,” *SIAM Journal on Scientific Computing*, vol. 42, no. 1, pp. A292–A317, 2020.
- [196] S. Cai, H. Li, F. Zheng, F. Kong, M. Dao, G. E. Karniadakis, and S. Suresh, “Artificial intelligence velocimetry and microaneurysm-on-a-chip for three-dimensional analysis of blood flow in physiology and disease,” *Proceedings of the National Academy of Sciences*, vol. 118, no. 13, p. e2100697118, 2021.
- [197] M. Cai, G. Karniadakis, and C. Li, “Fractional SEIR model and data-driven predictions of COVID-19 dynamics of omicron variant,” *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 32, no. 7, p. 071101, 2022.
- [198] S. Cai, Z. Wang, F. Fuest, Y. Jeon, C. Gray, and G. E. Karniadakis, “Flow over an espresso cup: inferring 3-d velocity and pressure fields from tomographic background oriented schlieren via physics-informed neural networks,” *Journal of Fluid Mechanics*, vol. 915, 2021.
- [199] K. Shukla, A. Jagtap, J. Blackshire, D. Sparkman, and G. E. Karniadakis, “A physics-informed neural network for quantifying the microstructural properties of polycrystalline nickel using ultrasound data: A promising approach for solving inverse problems,” *IEEE Signal Processing Magazine*, vol. 39, no. 1, pp. 68–77, 2022.
- [200] E. Zhang, M. Yin, and G. Karniadakis, “Physics-informed neural networks for nonhomogeneous material identification in elasticity imaging,” *arXiv:2009.04525v1 [cs.LG]*, 2020.
- [201] K. Nath, X. Meng, D. Smith, and G. Karniadakis, “Physics-informed neural networks for predicting gas flow dynamics and unknown parameters in diesel engines,” *Scientific Reports*, vol. 13, 2023.

- [202] W. Beitz, G. Pahl, and K. Grote, “Engineering design: a systematic approach,” *Mrs Bulletin*, vol. 71, 1996.
- [203] C. Cantwell, D. Moxey, A. Comerford, A. Bolis, G. Rocco, G. Mengaldo, D. De Grazia, S. Yakovlev, J.-E. Lombard, D. Ekelschot, *et al.*, “Nektar++: An open-source spectral/hp element framework,” *Computer Physics Communications*, vol. 192, pp. 205–219, 2015.
- [204] C. Geuzaine and J. Remacle, “Gmsh: A 3-d finite element mesh generator with built-in pre-and post-processing facilities,” *International Journal for Numerical Methods in Engineering*, vol. 79, no. 11, pp. 1309–1331, 2009.
- [205] C. Tsallis and D. Stariolo, “Generalized simulated annealing,” *Physica A: Statistical Mechanics and its Applications*, vol. 233, no. 1-2, pp. 395–406, 1996.
- [206] H. Alkaissi and S. McFarlane, “Artificial hallucinations in chatgpt: implications in scientific writing,” *Cureus*, vol. 15, 2023.
- [207] M. Sallam, “Chatgpt utility in healthcare education, research, and practice: systematic review on the promising perspectives and valid concerns,” in *Healthcare*, vol. 11, p. 887, MDPI, 2023.
- [208] R. Azamfirei, S. Kudchadkar, and J. Fackler, “Large language models and the perils of their hallucinations,” *Critical Care*, vol. 27, no. 1, pp. 1–2, 2023.
- [209] A. Ward, J. Liker, J. Cristiano, and D. K. Sobek II., “The second toyota paradox: How delaying decisions can make better cars faster,” *MIT Sloan Management Review*, 1995.
- [210] D. K. Sobek II., A. Ward, and J. Liker, “Toyota’s principles of set-based concurrent engineering,” *MIT Sloan Management Review*, 1999.

- [211] J. Royset, L. Bonfiglio, G. Vernengo, and S. Brizzolara, “Risk-adaptive set-based design and applications to shaping a hydrofoil,” *Journal of Mechanical Design*, vol. 139, no. 10, p. 101403, 2017.
- [212] B. Toche, R. Pellerin, and C. Fortin, “Set-based design: a review and new directions,” *Design Science*, vol. 6, p. e18, 2020.
- [213] L. Bonfiglio and J. Royset, “Multidisciplinary risk-adaptive set-based design of supercavitating hydrofoils,” *AIAA Journal*, vol. 57, no. 8, pp. 3360–3378, 2019.
- [214] K. Shintani, E. Nakatsugawa, and M. Tsuchiyama, “A set-based approach to dynamic system design using physics informed neural network,” *Journal of Advanced Mechanical Design, Systems, and Manufacturing*, vol. 16, no. 5, pp. JAMDSM0051–JAMDSM0051, 2022.
- [215] G. Wen, Z. Li, K. Azizzadenesheli, A. Anandkumar, and S. M. Benson, “U-fno—an enhanced fourier neural operator-based deep-learning model for multiphase flow,” *Advances in Water Resources*, vol. 163, p. 104180, 2022.
- [216] T. Tripura and S. Chakraborty, “Wavelet neural operator for solving parametric partial differential equations in computational mechanics problems,” *Computer Methods in Applied Mechanics and Engineering*, vol. 404, p. 115783, 2023.
- [217] Q. Cao, S. Goswami, and G. Karniadakis, “LNO: Laplace neural operator for solving differential equations,” *arXiv:2303.10528v2 [cs.LG]*, 2023.
- [218] O. Ovadia, A. Kahana, P. Stinis, E. Turkel, and G. Karniadakis, “ViTO: Vision transformer-operator,” *arXiv:2303.08891v1 [cs.CV]*, 2023.
- [219] K. Shukla, A. Jagtap, and G. Karniadakis, “Parallel physics-informed neural networks via domain decomposition,” *Journal of Computational Physics*, vol. 447, p. 110683, 2021.

- [220] A. Jagtap, K. Kawaguchi, and G. Karniadakis, “Adaptive activation functions accelerate convergence in deep and physics-informed neural networks,” *Journal of Computational Physics*, vol. 404, p. 109136, 2020.
- [221] S. Qian, H. Liu, C. Liu, S. Wu, and H. San Wong, “Adaptive activation functions in convolutional neural networks,” *Neurocomputing*, vol. 272, pp. 204–212, 2018.
- [222] G. Pahl, W. Beitz, J. Feldhusen, and K.-H. Grote, *Engineering Design: A Systematic Approach*, pp. 128–134. Springer London, 3 ed., 2007. Chapter 4.
- [223] L. Argote, “Organizational Forgetting,” in *Organizational Learning: Creating, Retaining and Transferring Knowledge*, ch. 3, pp. 57–83, New York: Springer, 2 ed., 2013.
- [224] D. J. Pauleen and G. E. Gorman, *Personal Knowledge Management: Individual, Organizational and Social Perspectives*, pp. 1–10. Gower Publishing, Ltd., 1 ed., 2012. Chapter 1.
- [225] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. F. Christiano, J. Leike, and R. Lowe, “Training language models to follow instructions with human feedback,” in *Advances in Neural Information Processing Systems* (S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, eds.), vol. 35, pp. 27730–27744, Curran Associates, Inc., 2022.
- [226] R. Heesch, S. Eilermann, A. Windmann, A. Diedrich, P. Rosenthal, and O. Nigemann, “Evaluating Large Language Models for Real-World Engineering Tasks,” *arXiv preprint arXiv:2505.13484*, 2025.
- [227] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom, “Toolformer: Language Models Can Teach

- Themselves to Use Tools,” *Advances in Neural Information Processing Systems*, vol. 36, pp. 68539–68551, 2023.
- [228] S. Yuan, K. Song, J. Chen, X. Tan, Y. Shen, K. Ren, D. Li, and D. Yang, “EASY-TOOL: Enhancing LLM-based Agents with Concise Tool Instruction,” in *ICLR 2024 Workshop on Large Language Model (LLM) Agents*, 2024.
- [229] F. Chiarello, S. Barandoni, M. M. Škec, and G. Fantoni, “Generative large language models in engineering design: opportunities and challenges,” *Proceedings of the Design Society*, vol. 4, pp. 1959–1968, 2024.
- [230] M. R. Cutkosky, R. S. Englemore, R. E. Fikes, M. R. Genesereth, T. R. Gruber, W. S. Mark, J. M. Tenenbaum, and J. C. Weber, “PACT: An experiment in integrating concurrent engineering systems,” *Computer*, vol. 26, no. 1, pp. 28–37, 1993.
- [231] H. Park, M. R. Cutkosky, A. B. Conru, and S.-H. Lee, “An agent-based approach to concurrent cable harness design,” *AI EDAM*, vol. 8, no. 1, pp. 45–61, 1994.
- [232] W. Shen and J.-P. A. Barthes, “DIDE: A Multi-Agent Environment for Engineering Design,” in *ICMAS*, vol. 95, pp. 12–14, 1995.
- [233] W. Shen and J.-P. A. Barthès, “An experimental multi-agent environment for engineering design,” *International Journal of Cooperative Information Systems*, vol. 5, no. 02n03, pp. 131–151, 1996.
- [234] H. Parunak, A. Ward, M. Fleischer, and J. Sauter, “A marketplace of design agents for distributed concurrent set-based design,” tech. rep., DARPA, 1997.
- [235] M. Campbell, J. Cagan, and K. Kotovsky, “Agent-based synthesis of electro-mechanical design configurations,” in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 80333, p. V003T03A003, American Society of Mechanical Engineers, 1998.



- [236] M. I. Campbell, J. Cagan, and K. Kotovsky, "A-design: an agent-based approach to conceptual design in a dynamic environment," *Research in Engineering Design*, vol. 11, no. 3, pp. 172–192, 1999.
- [237] J. Sun, Y. Zhang, and A. Nee, "Agent-based product design and planning for distributed concurrent engineering," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 4, pp. 3101–3106, 2000.
- [238] J. Sun, Y. F. Zhang, and A. Y. C. Nee, "A distributed multi-agent environment for product design and manufacturing planning," *International Journal of Production Research*, vol. 39, no. 4, pp. 625–645, 2001.
- [239] B. Lees, C. Branki, and I. Aird, "A framework for distributed agent-based engineering design support," *Automation in Construction*, vol. 10, no. 5, pp. 631–637, 2001.
- [240] G. Zhao, J. Deng, and W. Shen, "CLOVER: an agent-based approach to systems interoperability in cooperative design systems," *Computers in Industry*, vol. 45, no. 3, pp. 261–276, 2001.
- [241] C. Anumba, O. Ugwu, L. Newnham, and A. Thorpe, "A multi-agent system for distributed collaborative design," *Logistics Information Management*, vol. 14, no. 5/6, pp. 355–367, 2001.
- [242] K.-M. Chao, P. Norman, R. Anane, and A. James, "An agent-based approach to engineering design," *Computers in Industry*, vol. 48, no. 1, pp. 17–27, 2002.
- [243] C.-C. Huang, "A multi-agent approach to collaborative design of modular products," *Concurrent Engineering*, vol. 12, no. 1, pp. 39–47, 2004.
- [244] T. Madhusudan, "An agent-based approach for coordinating product design workflows," *Computers in Industry*, vol. 56, no. 3, pp. 235–259, 2005.

- [245] C.-H. Chu, P.-H. Wu, and Y.-C. Hsu, “Multi-agent collaborative 3D design with geometric model at different levels of detail,” *Robotics and Computer-Integrated Manufacturing*, vol. 25, no. 2, pp. 334–347, 2009.
- [246] E. Dostatni, J. Diakun, D. Grajewski, R. Wichniarek, and A. Karwasz, “Multi-agent system to support decision-making process in design for recycling,” *Soft Computing*, vol. 20, no. 11, pp. 4347–4361, 2016.
- [247] P. C. Gembarski, “Agent collaboration in a multi-agent-system for analysis and optimization of mechanical engineering parts,” *Procedia Computer Science*, vol. 176, pp. 592–601, 2020.
- [248] S. Plappert, P. C. Gembarski, and R. Lachmayer, “Knowledge-based design evaluation of rotational CAD-models with a multi-agent system,” in *International Conference On Systems Engineering*, pp. 47–56, Springer, 2021.
- [249] S. Plappert, P. C. Gembarski, and R. Lachmayer, “Development of a knowledge-based and collaborative engineering design agent,” *Procedia Computer Science*, vol. 207, pp. 946–955, 2022.
- [250] S. Plappert, C. Becker, P. C. Gembarski, and R. Lachmayer, “Scalable BDI-based Multi-Agent System for Digital Design Reviews,” *Procedia Computer Science*, vol. 225, pp. 3593–3602, 2023.
- [251] A. P. Gomez, P. Krus, M. Panarotto, and O. Isaksson, “Large language models in complex system design,” *Proceedings of the Design Society*, vol. 4, pp. 2197–2206, 2024.
- [252] L. Makatura, M. Foshey, B. Wang, F. Hähnlein, P. Ma, B. Deng, M. Tjandrasuwita, A. Spielberg, C. Owens, P. Y. Chen, *et al.*, “How can large language models help humans in design and manufacturing? Part 1: Elements of the LLM-enabled

- computational design and manufacturing pipeline,” *Harvard Data Science Review*, no. Special Issue 5, 2024.
- [253] L. Makatura, M. Foshey, B. Wang, F. Hähnlein, P. Ma, B. Deng, M. Tjandrasuwita, A. Spielberg, C. Owens, P. Y. Chen, *et al.*, “How can large language models help humans in design and manufacturing? Part 2: Synthesizing an end-to-end LLM-enabled design and manufacturing workflow,” *Harvard Data Science Review*, no. Special Issue 5, 2024.
- [254] C. Picard, K. M. Edwards, A. C. Doris, B. Man, G. Giannone, M. F. Alam, and F. Ahmed, “From concept to manufacturing: Evaluating vision-language models for engineering design,” *Artificial Intelligence Review*, vol. 58, no. 9, p. 288, 2025.
- [255] A. Schüpbach, R. San Miguel, J. Ferchow, and M. Meboldt, “From text to design: a framework to leverage LLM agents for automated CAD generation,” *Proceedings of the Design Society*, vol. 5, pp. 1893–1902, 2025.
- [256] M. J. Buehler, “Accelerating scientific discovery with generative knowledge extraction, graph-based representation, and multimodal intelligent graph reasoning,” *Machine Learning: Science and Technology*, vol. 5, no. 3, p. 035083, 2024.
- [257] M. A. Ferrag, N. Tihanyi, and M. Debbah, “From LLM Reasoning to Autonomous AI Agents: A Comprehensive Review,” *arXiv preprint arXiv:2504.19678*, 2025.
- [258] C. C. Obieke, J. Bridgeman, and J. Han, “A framework of AI collaboration in engineering design (AICED),” *Proceedings of the Design Society*, vol. 5, pp. 91–100, 2025.
- [259] I. M. Ilevbare, D. Probert, and R. Phaal, “A review of TRIZ, and its benefits and challenges in practice,” *Technovation*, vol. 33, no. 2, pp. 30–37, 2013.
- [260] J. R. Hauser and D. Clausing, “The house of quality,” *Harvard Business Review*, vol. 6, pp. 63–73, 1988.

- [261] NASA, “GoAero Prize.” <https://www.herox.com/goaero>, Jan 2025. Accessed: 10 Dec 2025.
- [262] N. Panta, S. Kafley, R. Acharya, S. Parajuli, D. Parajuli, P. Panta, S. Belbase, S. Pant, A. Regmi, A. Tanaka, *et al.*, “MEDA: A Multi-Agent System For Parametric CAD Model Creation,” in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 89237, p. V03BT03A042, American Society of Mechanical Engineers, 2025.
- [263] S. Chen, A. E. Bayrak, and Z. Sha, “A Cost-Aware Multi-Agent System for Black-Box Design Space Exploration,” *Journal of Mechanical Design*, vol. 147, no. 1, p. 011703, 2025.
- [264] P. Ghasemi and M. Moghaddam, “Vision-Language Models for Design Concept Generation: An Actor–Critic Framework,” *Journal of Mechanical Design*, vol. 147, no. 9, p. 091402, 2025.
- [265] S. Massoudi and M. Fuge, “Agentic large language models for conceptual systems engineering and design,” in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 89237, p. V03BT03A045, American Society of Mechanical Engineers, 2025.
- [266] Z. Wang, F. P. W. Lo, Q. Chen, Y. Zhang, C. Lin, X. Chen, Z. Yu, A. J. Thompson, E. M. Yeatman, and B. P. Lo, “An LLM-enabled Multi-Agent Autonomous Mechatronics Design Framework,” in *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 4205–4215, 2025.
- [267] S. Jiang, M. Xie, F. Y. Chen, J. Ma, and J. Luo, “Intelligent Design 4.0: Paradigm Evolution Toward the Agentic Artificial Intelligence Era,” *Journal of Computing and Information Science in Engineering*, vol. 25, no. 12, p. 120808, 2025.

- [268] S. Lander, “Issues in multiagent design systems,” *IEEE Expert*, vol. 12, no. 2, pp. 18–26, 1997.
- [269] P. N. Stuart J. Russell, “Logical Agents,” in *Artificial Intelligence: A Modern Approach*, ch. 7, pp. 226–264, New York: Pearson, 4 ed., 2021.
- [270] C. Sun, S. Huang, and D. Pompili, “LLM-Based Multi-Agent Decision-Making: Challenges and Future Directions,” *IEEE Robotics and Automation Letters*, vol. 10, no. 6, pp. 5681–5688, 2025.
- [271] R. E. Bilstein, “NACA Origins (1915-1930),” in *Orders of Magnitude: A History of the NACA and NASA, 1915-1990*, vol. 4406, ch. 1, pp. 10–12, National Aeronautics and Space Administration, 1989.
- [272] I. H. Abbot and A. E. V. Doenhoff, “Families of Wing Sections,” in *Theory of Wing Sections: Including a Summary of Airfoil Data*, ch. 6, pp. 111–120, Dover, 2012.
- [273] I. H. Abbot and A. E. V. Doenhoff, “Aerodynamic Characteristics of Wing Sections,” in *Theory of Wing Sections: Including a Summary of Airfoil Data*, ch. Appendix IV, Dover, 2012.
- [274] W. G. Sewall, R. J. McGhee, J. K. Viken, E. G. Waggoner, B. S. Walker, and B. F. Millard, “Wind tunnel results for a high-speed, natural laminar-flow airfoil designed for general aviation aircraft,” tech. rep., NASA, 1985.
- [275] P. D. Sharpe, *Accelerating Practical Engineering Design Optimization with Computational Graph Transformations*. PhD thesis, Massachusetts Institute of Technology, 2024. Available at <https://dspace.mit.edu/handle/1721.1/157809>.
- [276] P. Sharpe and R. J. Hansman, “NeuralFoil: An Airfoil Aerodynamics Analysis Tool Using Physics-Informed Machine Learning,” *arXiv preprint arXiv:2503.16323*, 2025.

- [277] F. M. Suchanek, G. Kasneci, and G. Weikum, “Yago: A Core of Semantic Knowledge,” in *Proceedings of the 16th international conference on World Wide Web*, pp. 697–706, 2007.
- [278] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann, “DBpedia - A crystallization point for the Web of Data,” *Journal of Web Semantics*, vol. 7, no. 3, pp. 154–165, 2009.
- [279] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, “Freebase: a collaboratively created graph database for structuring human knowledge,” in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pp. 1247–1250, Association for Computing Machinery, 2008.
- [280] E. Rajabi and K. Etminani, “Knowledge-graph-based explainable AI: A systematic review,” *Journal of Information Science*, vol. 50, no. 4, pp. 1019–1029, 2024.
- [281] LangChain, “LangChain text splitters,” 2025. Last accessed: Nov 6, 2025.
- [282] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,” in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, eds.), vol. 33, pp. 9459–9474, Curran Associates, Inc., 2020.
- [283] N. Reimers and I. Gurevych, “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks,” *arXiv preprint arXiv:1908.10084*, 2019.
- [284] B. M. Kulfan, “Universal Parametric Geometry Representation Method,” *Journal of Aircraft*, vol. 45, no. 1, pp. 142–158, 2008.
- [285] D. G. Ullman, *The mechanical design process*, ch. 1, pp. 3–7. McGraw-Hill Companies, Inc, 2010.

- [286] D. G. Jansson and S. M. Smith, “Design fixation,” *Design studies*, vol. 12, no. 1, pp. 3–11, 1991.
- [287] D. K. Sobek II, A. C. Ward, and J. K. Liker, “Toyota’s principles of set-based concurrent engineering,” *MIT sloan management review*, 1999.
- [288] D. J. Singer, N. Doerry, and M. E. Buckley, “What is set-based design?,” *Naval Engineers Journal*, vol. 121, no. 4, pp. 31–43, 2009.
- [289] S. Kaplan and B. J. Garrick, “On the quantitative definition of risk,” *Risk Analysis*, vol. 1, no. 1, pp. 11–27, 1981.
- [290] M. C. Kennedy and A. O’Hagan, “Bayesian calibration of computer models,” *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 63, no. 3, pp. 425–464, 2001.
- [291] G. A. Hazelrigg, “A framework for decision-based engineering design,” *Journal of Mechanical Design*, vol. 120, pp. 653–658, 12 1998.
- [292] W. Oberkampf and J. Helton, “Investigation of evidence theory for engineering applications,” in *43rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, p. 1569, 2002.
- [293] N. M. Alexandrov, R. M. Lewis, C. R. Gumbert, L. L. Green, and P. A. Newman, “Approximation and model management in aerodynamic optimization with variable-fidelity models,” *Journal of Aircraft*, vol. 38, no. 6, pp. 1093–1101, 2001.
- [294] T. W. Simpson, J. D. Poplinski, P. N. Koch, and J. K. Allen, “Metamodels for computer-based engineering design: survey and recommendations,” *Engineering with Computers*, vol. 17, no. 2, pp. 129–150, 2001.
- [295] W. L. Oberkampf, J. C. Helton, C. A. Joslyn, S. F. Wojtkiewicz, and S. Ferson, “Challenge problems: uncertainty in system response given uncertain parameters,”

- Reliability Engineering & System Safety*, vol. 85, no. 1, pp. 11–19, 2004. Alternative Representations of Epistemic Uncertainty.
- [296] R. L. Keeney and H. Raiffa, *Decisions with Multiple Objectives: Preferences and Value Trade-Offs*. Cambridge University Press, 1993.
  - [297] W. J. Morokoff and R. E. Caflisch, “Quasi-monte carlo integration,” *Journal of Computational Physics*, vol. 122, no. 2, pp. 218–230, 1995.
  - [298] S. Pandey, R. Xu, W. Wang, and X. Chu, “OpenFOAMGPT: A retrieval-augmented large language model (LLM) agent for OpenFOAM-based computational fluid dynamics,” *Physics of Fluids*, vol. 37, no. 3, 2025.
  - [299] L. Lu, P. Jin, and G. E. Karniadakis, “Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators,” *arXiv preprint arXiv:1910.03193*, 2019.
  - [300] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *Medical image computing and computer-assisted intervention—MICCAI 2015: 18th international conference, Munich, Germany, October 5–9, 2015, proceedings, part III 18*, pp. 234–241, Springer, 2015.
  - [301] J. K. Gupta and J. Brandstetter, “Towards multi-spatiotemporal-scale generalized PDE modeling,” *Transactions on Machine Learning Research*, 2023.
  - [302] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. Battaglia, “Learning to simulate complex physics with graph networks,” in *International Conference on Machine Learning*, pp. 8459–8468, PMLR, 2020.
  - [303] A. Krizhevsky, G. Hinton, *et al.*, “Convolutional deep belief networks on cifar-10,” *Unpublished manuscript*, vol. 40, no. 7, pp. 1–9, 2010.
  - [304] ASME, *A New Framework for Collaborative Set-Based Design: Application to the Design Problem of a Hollow Cylindrical Cantilever Beam*, vol. Volume 5: 37th



Design Automation Conference, Parts A and B of *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, 08 2011.

- [305] S. Hannapel and N. Vlahopoulos, “Implementation of set-based design in multidisciplinary design optimization,” *Structural and Multidisciplinary Optimization*, vol. 50, no. 1, pp. 101–112, 2014.
- [306] A. Riaz, M. D. Guenov, and A. Molina-Cristobal, “Set-based approach to passenger aircraft family design,” *Journal of Aircraft*, vol. 54, no. 1, pp. 310–326, 2017.
- [307] C. Small, R. Buchanan, E. Pohl, G. S. Parnell, M. Cilli, S. Goerger, and Z. Wade, “A uav case study with set-based design,” *INCOSE International Symposium*, vol. 28, no. 1, pp. 1578–1591, 2018.
- [308] E. Specking, G. Parnell, E. Pohl, and R. Buchanan, “Early design space exploration with model-based system engineering and set-based design,” *Systems*, vol. 6, no. 4, 2018.
- [309] Z. Wade, G. S. Parnell, S. R. Goerger, E. Pohl, and E. Specking, “Designing engineered resilient systems using set-based design,” in *Systems Engineering in Context: Proceedings of the 16th Annual Conference on Systems Engineering Research*, pp. 111–122, Springer, 2019.
- [310] T. A. McKenney, L. F. Kemink, and D. J. Singer, “Adapting to changes in design requirements using set-based design,” *Naval Engineers Journal*, vol. 123, no. 3, pp. 67–77, 2011.
- [311] A. Georgiades, S. Sharma, T. Kipouros, and M. Savill, “Adopt: An augmented set-based design framework with optimisation,” *Design Science*, vol. 5, p. e4, 2019.
- [312] A. T. Beck and W. J. de Santana Gomes, “A comparison of deterministic, reliability-based and risk-based structural optimization under uncertainty,” *Probabilistic Engi-*

- neering Mechanics*, vol. 28, pp. 18–29, 2012. Computational Stochastic Mechanics — CSM6.
- [313] A. Chaudhuri, M. Norton, and B. Kramer, “Risk-based design optimization via probability of failure, conditional value-at-risk, and buffered probability of failure,” in *AIAA Scitech 2020 Forum*, p. 2130, AIAA, 2020.
  - [314] A. Chaudhuri, B. Kramer, M. Norton, J. O. Royset, and K. Willcox, “Certifiable risk-based engineering design optimization,” *AIAA Journal*, vol. 60, no. 2, pp. 551–565, 2022.
  - [315] M. P. Rumpfkeil, “Robust design under mixed aleatory/epistemic uncertainties using gradients and surrogates,” *Journal of Uncertainty Analysis and Applications*, vol. 1, no. 1, p. 7, 2013.
  - [316] W. Li, M. Xiao, A. Garg, and L. Gao, “A new approach to solve uncertain multidisciplinary design optimization based on conditional value at risk,” *IEEE Transactions on Automation Science and Engineering*, vol. 18, no. 1, pp. 356–368, 2021.
  - [317] W. Li, C. Li, L. Gao, and M. Xiao, “Risk-based design optimization under hybrid uncertainties,” *Engineering with Computers*, vol. 38, no. 3, pp. 2037–2049, 2022.
  - [318] L. Padovan, V. Pediroda, and C. Poloni, “Multi objective robust design optimization of airfoils in transonic field,” in *Multidisciplinary Methods for Analysis Optimization and Control of Complex Systems*, pp. 283–295, Springer, 2005.
  - [319] J. Gottweis, W.-H. Weng, A. Daryin, T. Tu, A. Palepu, P. Sirkovic, A. Myaskovsky, F. Weissenberger, K. Rong, R. Tanno, *et al.*, “Towards an ai co-scientist,” *arXiv preprint arXiv:2502.18864*, 2025.
  - [320] K. Swanson, W. Wu, N. L. Bulaong, J. E. Pak, and J. Zou, “The virtual lab of ai agents designs new sars-cov-2 nanobodies,” *Nature*, vol. 646, no. 8085, pp. 716–723, 2025.

- [321] P. Ramachandran, B. Zoph, and Q. V. Le, “Searching for activation functions,” *arXiv preprint arXiv:1710.05941*, 2017.
- [322] X. Chen, C. Liang, D. Huang, E. Real, K. Wang, H. Pham, X. Dong, T. Luong, C.-J. Hsieh, Y. Lu, and Q. V. Le, “Symbolic discovery of optimization algorithms,” in *Advances in Neural Information Processing Systems* (A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, eds.), vol. 36, pp. 49205–49233, Curran Associates, Inc., 2023.