UCLA
UCLA Electronic Theses and Dissertations

Title

Foundation Models for Partial Differential Equations: Learning Spatiotemporal Dynamics at Scale

Permalink

https://escholarship.org/uc/item/0n15p678

ISBN

9798247907626

Author

Liu, Yuxuan

Publication Date

2026-05-22

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA
Los Angeles

Foundation Models for Partial Differential Equations: Learning Spatiotemporal Dynamics at Scale

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Mathematics

by

Yuxuan Liu

2026

© Copyright by
Yuxuan Liu
2026

ABSTRACT OF THE DISSERTATION

Foundation Models for Partial Differential Equations:
Learning Spatiotemporal Dynamics at Scale

by

Yuxuan Liu
Doctor of Philosophy in Mathematics
University of California, Los Angeles, 2026
Professor Hayden Schaeffer, Chair

Modeling spatiotemporal dynamics governed by partial differential equations (PDEs) remains a fundamental challenge in computational physics, requiring machine learning models that can generalize across diverse physical systems, parameter ranges, geometries, and temporal scales. This thesis focuses on the development of scalable PDE foundation models through three novel transformer-based architectures. First, we introduce PROSE-FD, a multimodal foundation model that fuses symbolic system information with trajectory data to enable zero-shot, non-autoregressive operator learning. Second, we present Vision In-Context Operator Networks (VICON), which leverage contextual information to be a few-shot learner for multi-physics systems, demonstrating strong robustness to imperfect or missing measurements. Finally, we propose BCAT, a block causal transformer for autoregressive prediction that effectively captures complex spatial dependencies by utilizing previous frames as contextual priors. To ground this approach, we provide a rigorous theoretical analysis establishing BCAT as a universal approximator for dynamical system solution operators. Together, these frameworks provide a diverse set of methodologies for learning and predicting complex

spatiotemporal systems, spanning multimodal operator learning, in-context learning, and causal autoregression. These models advance the state-of-the-art in PDE modeling and lay the foundation for future research in scalable, generalizable models for scientific computing.

The dissertation of Yuxuan Liu is approved.

Chenfanfu Jiang

Guido F. Montúfar Cuartas

Deanna M. Needell

Hayden Schaeffer, Committee Chair

University of California, Los Angeles

2026

To Curiosity.

TABLE OF CONTENTS

1 Introduction **1**

2 PROSE-FD: A Multimodal PDE Foundation Model for Learning Multiple Operators for Forecasting Fluid Dynamics **4**

2.1 Introduction 4

2.2 Related Works 7

2.2.1 Operator Learning 7

2.2.2 PDE Foundation Models 7

2.2.3 Multimodal Machine Learning 8

2.3 Methods 9

2.3.1 Problem Setting 9

2.3.2 Model Overview 9

2.3.3 Patch-based Data Encoder 10

2.3.4 Equation Encoding and Fusion 11

2.3.5 Patch-based Operator Decoder 11

2.4 Experiments 13

2.4.1 Experiment Setup 13

2.4.2 Baselines and Comparisons 14

2.4.3 Main Results 14

2.4.4 Ablation Studies 15

2.4.5 Resource Usage Comparison 19

2.5 Conclusion 20

2.A Dataset Details 20
2.A.1 PDEBench 20
2.A.2 PDEArena 21
2.A.3 CFDBench 22
2.B Experiment Details 22
2.B.1 Training 22
2.B.2 Model Hyperparameters 23
2.B.3 Baselines 23
2.B.4 More Visualizations 24

3 VICON: Vision In-Context Operator Networks for Multi-Physics Fluid Dynamics Prediction 27

3.1 Introduction 27
3.2 Related Work 30
3.3 Preliminaries 31
3.4 Methodology 33
3.4.1 Problem Setup 33
3.4.2 Vision In-Context Operator Networks 34
3.4.3 Prompt Normalization 36
3.4.4 Datasets and Data Augmentation 37
3.4.5 Inference with Flexible Strategies 38
3.4.6 Evaluation Metric 39
3.5 Experimental Results 39
3.5.1 Superior Performance on Long-term Rollout Predictions 41

3.5.2 Flexible Rollout Strategies 42
3.5.3 Importance of In-Context Learning 43
3.5.4 Robustness to Imperfect Temporal Measurements 44
3.5.5 Turbulence Kinetic Energy Analysis 45
3.5.6 Benefits of Multi-Physics Joint Training 45
3.5.7 Context Sampling Strategy 46
3.5.8 Computational Efficiency 46
3.6 Conclusion and Future Work 47
3.A Dataset Details 49
3.A.1 PDEArena-Incomp Dataset 49
3.A.2 PDEBench-Comp-HighVis and PDEBench-Comp-LowVis Datasets . 50
3.A.3 QOI Union and Channel Mask 50
3.B Experiment Details 51
3.B.1 VICON Model Details 51
3.B.2 Training Details 51
3.B.3 MPP Details 51
3.B.4 DPOT Details 53
3.B.5 Specialist U-Net Details 53
3.C Algorithms and Examples of Strategy Generation 54
3.C.1 Strategy with Full Temporal Sequence 54
3.C.2 Strategy with Imperfect Temporal Sequence 55
3.D Additional Experimental Results. 56
3.D.1 Ablation Studies 56

3.D.2 Analysis of Checkerboard Artifacts in PDEBench-Comp-HighVis . . . 59
3.D.3 Cross-Trajectory Context Sampling . . . 60
3.D.4 More Results and Visualizations . . . 60

4 BCAT: A Block Causal Transformer for PDE Foundation Models for Fluid Dynamics . . . 79

4.1 Introduction . . . 79
4.2 Related Works . . . 81
4.2.1 PDE Foundation Models . . . 81
4.2.2 Transformers for Time Series . . . 82
4.3 Methods . . . 82
4.3.1 Problem Setup . . . 82
4.3.2 Model Overview . . . 83
4.3.3 Image Tokenization via Patch Embedding . . . 83
4.3.4 Preliminary: Next Token Prediction . . . 84
4.3.5 Next Frame Prediction . . . 85
4.3.6 Muon Optimizer . . . 87
4.3.7 Implementation Details . . . 88
4.4 Experiments . . . 88
4.4.1 Experiment Setup . . . 88
4.4.2 Main Results . . . 90
4.4.3 Transfer Capability . . . 92
4.4.4 Ablation Studies . . . 93
4.4.5 Resource Usage Comparison . . . 98

4.4.6 Experimental Summary and Model Novelty 98
4.5 Theoretical Analysis of BCAT 100
4.5.1 Notations and Preliminary Results 101
4.5.2 Overview 102
4.5.3 Lemmata 103
4.5.4 Main Approximation Theorem 108
4.5.5 Approximation of Dynamical System Solution Operators 110
4.6 Conclusion 112
4.A Dataset Details 113
4.A.1 PDEBench 113
4.A.2 PDEArena 114
4.A.3 CFDBench 115
4.B Experiment Details 115
4.B.1 Training 115
4.B.2 Model Hyperparameters 116
4.B.3 Baselines and Comparisons 116
4.B.4 Necessity of Retraining for Fairness 118
4.B.5 Remarks on Single-Task Models 119
4.C Visualizations 119

5 Conclusion 127

References 129

LIST OF FIGURES

2.1 **PROSE-FD model overview.** The inputs to the model are the trajectories and governing equations (in symbolic form) sampled from the datasets. The input data is patchified before being converted into a sequence of features, which are then processed with encoders and fused with processed symbolic information. The data decoder takes in fused features and generates predictions at given query locations. 6

2.2 **Two example outputs for the PROSE-FD model.** 17

2.3 **Comparing PROSE-FD trained with Muon vs. AdamW optimizer.** The target (first row) is the first 6 output steps from the PDEArena Navier-Stokes (conditioned) dataset (y-velocity channel). For each optimizer (each row), we display the difference between the target and model output. Relative L^2 errors for the full trajectories are listed after the optimizer names. 19

2.4 **Example outputs for the PROSE-FD model.** 6 consecutive output steps for PDEArena NS-cond dataset (all three channels in equation (2.13)). Each column represents a different timestep. For this trajectory, the relative L^2 error is 7.79%. This example shows that even when the relative error is higher, the structures of the flow can still be predicted correctly. 25

2.5 **Comparing outputs from different models.** The target (first row) is the first 6 output steps from the PDEBench Compressible Navier-Stokes dataset (x-velocity). For each model (each row), we display the difference between the target and model output. Relative L^2 errors for the full trajectories are listed after the model names. 26

3.1 **VICON model overview.** (a) The formation process for conditions (COND) and quantities of interest (QOI) pairs. Δt is randomly sampled during training. (b) Model illustration. The inputs to the model are pairs of COND and QOI, which are patchified and flattened before feeding into the transformer layers. The outputs, which represent different patches in the output frame, are transformed back to obtain the final predictions. (c) With imperfect temporal measurements, VICON forms pairs using only clean data, and does not need to fill missing frames. 32

3.2 **Main experiment results.** (a) Last step rollout errors on 3 datasets. VICON outperforms MPP on all datasets and outperforms DPOT on 2 datasets. (b) VICON allows flexible rollout strategies to reduce error accumulation and demonstrates stride extrapolation. (c) VICON is robust to imperfect temporal measurements, while MPP and DPOT suffer from performance degradation. (d) VICON is smaller in size and has faster rollout time per step. 40

3.3 **Comparison of rollout errors (scaled by std) across different datasets and models.** We show errors for two VICON rollout strategies: single step rollout and flexible rollout strategy. For flexible step, step 3 works optimally for the PDEArena-Incomp dataset, while step 1 works best for PDEBench-Comp-LowVis and PDEBench-Comp-HighVis. 42

3.4 Ablation studies across the three datasets. **Top row (a-c):** Impact of patch resolutions $(4, 8, 16, 32, 64)$ showing optimal performance at patch size 16. **Middle row (d-f):** Effect of different positional encoding combinations. **Bottom row (g-i):** Performance variation with different context lengths (6, 8, 10 pairs). . . . 57

3.5 Comparison of rollout errors across different datasets, using single-step and flexible-step strategies with varying maximum step sizes ($s_{\max} = 1, 3, 5, 7$). 61

3.6 **Comparison of turbulence kinetic energy predictions.** (Top-left) Ground truth TKE field, (Top-right) model predictions, (Bottom) model error. 61

3.7 **Comparing rollout errors (single step, scale by std) for joint versus separate training strategies**. For separate training (individual models for each dataset), we maintain the same batch sizes as in joint training while adjusting training steps to be slightly more than one-third of the joint training duration, ensuring comparable total computational costs across both approaches. 62

3.8 **Comparison of rollout errors (scale by std) with different input data noise levels across all datasets.** Each column (left to right) shows VICON, DPOT, and MPP models, while each row (top to bottom) represents PDEBench-Comp-LowVis, PDEBench-Comp-HighVis, and PDEArena-Incomp datasets. For MPP and DPOT which require fixed dt and context window, interpolation is used to generate missing frames, while VICON can directly handle irregular temporal data without interpolation. 63

3.9 **Comparing outputs from different models.** The target is the first 6 output steps from PDEBench-Comp-HighVis dataset (x-velocity channel). For each model (each row), we display the difference between target and model output. Errors (rescaled by std) for the full trajectory are listed after the model names. 64

3.10 **Comparing outputs from different models.** The target is the first 6 output steps from PDEBench-Comp-LowVis dataset (pressure channel). For each model (each row), we display the difference between target and model output. Errors (rescaled by std) for the full trajectory are listed after the model names. 65

3.11 **Comparing outputs from different models.** The target is the first 6 output steps from PDEArena-Incomp dataset (particle density channel). For each model (each row), we display the difference between target and model output. Errors (rescaled by std) for the full trajectory are listed after the model names. 66

3.12 **Example outputs for the VICON model.** (a) The pressure field of PDEBench-Comp-HighVis dataset, and (b) the x-velocity field of PDEArena-Incomp dataset. Each column represents a different timestep. 67

3.13 **More example outputs for the VICON model.** Showing 5 output steps for the PDEBench-Comp-LowVis dataset as governed by equation (3.9): (a) y-velocity, (b) density, and (c) pressure fields. Each column represents a different timestep. 68

4.1 **BCAT model overview.** The inputs to the model are the initial frames sampled from the datasets, which are patchified and converted into a sequence of features. Transformer layers then take the input sequence to perform next frame prediction, where a block causal mask allows spatial interactions within a frame and temporal causality across varying-length context windows. The processed features are then transformed back to form the final predictions. 121

4.2 **Evaluating BCAT, DPOT-L, and MPP-L for more output time steps on PDEArena NS-cond dataset.** Rollout is used to obtain outputs beyond the training timesteps. 122

4.3 **Comparing BCAT trained with Muon vs. AdamW optimizer.** The target (first row) is the first 6 output steps from the PDEArena Navier-Stokes (conditioned) dataset (particle density channel). For each optimizer (each row), we display the difference between the target and model output. Relative L^2 errors for the full trajectories are listed after the optimizer names. 122

4.4 Transformer Layers used in BCAT model. 123

4.5 **Example outputs from the BCAT model.** 4 output steps for PDEArena Navier-Stokes dataset. The channel plotted is the particle density in equation (4.44). Each column represents a different timestamp. For this trajectory, the relative L^2 error is 1.93%. 123

4.6 **Two example outputs from the BCAT model.** 124

4.7 **Comparing outputs from different models.** The target (first row) is the first 6 output steps from the PDEArena Navier-Stokes (conditioned) dataset (particle density channel). For each model (each row), we display the difference between the target and model output. Relative L^2 errors for the full trajectories are listed after the model names. 125

4.8 **Comparing outputs from different models.** The target is the first 6 output steps from the PDEBench Compressible Navier-Stokes dataset (y-velocity channel). For each model (each row), we display the difference between target and model output. Relative L^2 errors for the full trajectory are listed after the model names. 126

LIST OF TABLES

2.1 **Main Results and Comparisons with Baselines**. The numbers reported are relative L^2 errors (%). The averages are taken with respect to the 6 distinct families listed in the columns of the table. For each family of equations, we **bold** the best results. *Note that the PDEBench CNS contains 8 subsets of parameter configurations. 12

2.2 **Compare PROSE-FD trained with AdamW and Muon optimizer.** The numbers reported are relative L^2 errors (%). The averages are taken with respect to the 6 families. 13

2.3 **Results for PROSE-FD Ablation Studies.** We compare the baseline (operator) approach to a rollout (recursive) prediction variant and a data-only (no symbolic information) variant. The numbers reported are relative L^2 errors (%). The averages are taken with respect to the 6 distinct families listed in the columns of the table. 15

2.4 **Comparing the inference speed and GPU memory usage for all models.** We measure the recourse usage for different models and variants to generate 10 output steps for a single trajectory. 16

2.5 **Model hyperparameters.** FFN means feedforward network. 23

3.1 **Summary of Rollout Relative L^2 Error (scaled by std)** across different methods and datasets. The best results are highlighted in bold. For flexible step rollout, step 3 works best for the PDEArena-Incomp dataset. Specialist U-Net is trained individually per dataset with instance normalization matching VICON's preprocessing. 41

3.2 Non-in-context ViT baseline comparison on PDEArena-Incomp dataset. 44

3.3 Impact of incorrect context pairs on VICON performance. 44

3.4 Model Configuration Details. 52
3.5 Optimization Hyperparameters. 53
3.6 Single-step Rollout Strategy Example 70
3.7 Flexible-step Rollout Strategy Example ($s_{\max} = 5$) 70
3.8 Single-step Rollout Strategy Example with Missing Frames 74
3.9 Flexible-step Rollout Strategy Example with Missing Frames (max stride: 3) . . 74
3.10 CNN-ICON vs VICON comparison (normalized RMSE, scaled by std) 75
3.11 Standard deviation analysis across time phases for PDEBench datasets. The bolded velocity channels show PDEBench-Comp-HighVis has extreme variation between early and late phases, unlike PDEBench-Comp-LowVis with nearly consistent variation. This dynamic difference makes training a uniformed model particularly challenging. 75
3.12 Cross-trajectory context experiment on PDEArena-Incomp (normalized RMSE, scaled by std) 76
3.13 **(Comparison) Summary of Rollout Relative L^2 Error (scale by std)** for different methods across various cases. The best results are highlighted in bold. 76
3.14 **(Comparison) Summary of Rollout Absolute L^2 Error** for different methods across various cases. The best results are highlighted in bold. 77
3.15 **Summary of Rollout Relative L^2 Error Metrics (single step, scale by std) for 2 Training Strategies**. The best results are highlighted in bold. For separate training (a model on each dataset), each run's batch size is controlled to be the same as joint training. To ensure fair comparison, we maintain the same batch sizes as joint training for each separate training run while adjusting their training steps to be slightly more than one-third of the joint training duration, ensuring comparable total computational costs. 78

3.16 **(Comparison) Summary of Resource and Timing Metrics** for different methods. 78

4.1 **Main Results and Comparisons with Baselines**. The numbers reported are relative L^2 errors (%). The averages are taken with respect to the 6 distinct families listed in the columns of the table. We **bold** the best result in each column. 91

4.2 **Comparing Transfer Capabilities on PDEBench Turbulence dataset.** Each model is finetuned for 500 gradient updates with a batch size of 64 (20 epochs). 93

4.3 **Compare BCAT model trained with AdamW and Muon optimizer.** The numbers reported are relative L^2 errors (%). The averages are taken with respect to the 6 distinct families listed in the columns of the table. 94

4.4 **Comparing next frame prediction and next token prediction.** The errors shown are relative L^2 errors (%) for PDEArena NS dataset. *The Next-Token error is computed assuming all previous tokens are exactly known (i.e., the full trajectory is inputed to the model, similar to training time). It is not a standard metric and is only used to provide a comparison. 95

4.5 **Comparing the performance of BCAT and Time-then-Space Model.** The errors shown are relative L^2 errors (%). 96

4.6 **Comparing the performance of BCAT and ViT with different patch size.** The errors shown are relative L^2 errors (%). 97

4.7 **Additional Ablation Studies.** Each row represents a different variant and we underline the difference with the BCAT model (last row). The errors shown are relative L^2 errors (%). *The training loss diverges when using Muon without QK-Norm, thus for this row only, we report the result when trained using AdamW. 98

4.8 **Comparing the inference speed and GPU memory usage for all models.** We measure the resource usage for different models and variants to generate 10 output steps for a single trajectory. 99

4.9 **Model hyperparameters.** FFN means feedforward network. 116

4.10 **Evaluating Pretrained Checkpoints vs. Retrained Model.** * means directly evaluating released pretrained checkpoints. Numbers in () mean that the model is not pretrained on this dataset. The numbers reported are relative L^2 errors (%). The averages are taken with respect to the 6 distinct families listed in the columns of the table. 119

4.11 **Single-tasked models trained on all data simultaneously v.s. on each class separately.** * means the results are obtained by training separate models on each dataset. The numbers reported are relative L^2 errors (%). 120

ACKNOWLEDGMENTS

First and foremost, I would like to express my gratitude to my advisor, Prof. Hayden Schaeffer, for his support, guidance, and mentorship during both my undergraduate and graduate studies. I first learned about Prof. Schaeffer's research in the summer of 2020, and I was immediately drawn to his work on machine learning for scientific computing. Over the next two years at CMU and the following four years at UCLA, we worked on a wide range of projects, from random feature models for interacting agents to PDE foundation models for spatiotemporal dynamics. His mentorship has been instrumental in shaping my research direction and approach, and I am deeply grateful for his support and encouragement throughout my academic journey.

I would also like to thank my committee members, Prof. Chenfanfu Jiang, Prof. Guido Mont'ufar, and Prof. Deanna Needell, for their valuable feedback and constructive suggestions. Their expertise and perspectives have greatly enriched my work and helped refine my ideas.

During my undergraduate studies at CMU, I had the privilege of working with several exceptional mentors. I am deeply grateful to my first research mentor, Prof. William Hrusa, who sadly passed away before the completion of this thesis. His support, wisdom, and enthusiasm for research had a lasting impact on both this work and my development as a scholar. I also thank Prof. Ian Tice for guiding me into the field of analysis and PDEs, and for his support and encouragement during my undergraduate studies.

To my collaborators: Dr. Jingmin Sun, Prof. Zecheng Zhang, Jacob Murri, Prof. Scott McCalla, Prof. Liu Yang, Dr. Elisa Negrini, Prof. Stanley Osher, Dr. Yadi Cao, Prof. Rose Yu, Xinjie He, Griffin Pinney, Dr. Minxin Zhang, Dr. Yuning Shen, Nima Shoghi, Dr. Rob Brekelmans, Prof. Quanquan Gu, Dr. Daniel Gurevich, Tracy Yu, and Dr. Boonam Shin – I am grateful for your insights, expertise, and dedication, which have been invaluable to my research and growth as a researcher. I am also thankful for the opportunity to collaborate

with such talented individuals and for the friendships that have developed along the way.

I would like to thank the UCLA Department of Mathematics for providing a supportive research environment. My graduate studies at UCLA have been enriched by a vibrant community of scholars and available resources, and I am grateful for the opportunities to learn and grow as a graduate student. I also thank the funding sources that supported this research, including NSF CAREER Award DMS-2331100, NSF DMS-2331033, NSF DMS-2427558, NSF DMS-2502561, and AFOSR MURI FA9550-21-1-0084.

Most importantly, I would like to thank my family and my girlfriend Xinyue Li, for their unwavering support and encouragement throughout my academic journey. Your love, patience, and understanding have been a constant source of strength and motivation, and I am deeply grateful for everything you have done for me.

VITA

2018–2022 B.S. (Mathematical Sciences) and B.S. (Computer Science), Carnegie Mellon University.

2018–2022 M.S. (Mathematical Sciences), Carnegie Mellon University.

2022–2024 M.A. (Mathematics), University of California, Los Angeles.

2022–2026 Teaching and Research Assistant, Department of Mathematics, University of California, Los Angeles.

PUBLICATIONS

Minxin Zhang, **Yuxuan Liu**, and Hayden Schaeffer. Adam Improves Muon: Adaptive Moment Estimation with Orthogonalized Momentum. *arXiv preprint arXiv:2602.17080*, 2026

Nima Shoghi, **Yuxuan Liu**, Yuning Shen, Rob Brekelmans, Pan Li, and Quanquan Gu. Scalable SpatioTemporal SE(3) Diffusion for Long-Horizon Protein Dynamics. *International Conference on Learning Representations*, 2026

Yadi Cao*, **Yuxuan Liu***, Liu Yang, Rose Yu, Hayden Schaeffer, and Stanley Osher. VICON: Vision In-Context Operator Networks for Multi-Physics Fluid Dynamics Prediction. *Transactions on Machine Learning Research*, 2026

Minxin Zhang, **Yuxuan Liu**, and Hayden Schaeffer. AdaGrad Meets Muon: Adaptive Stepsizes for Orthogonal Updates. *NeurIPS Optimization for ML Workshop*, 2025

Elisa Negrini, **Yuxuan Liu**, Liu Yang, Stanley J Osher, and Hayden Schaeffer. A Multimodal PDE Foundation Model for Prediction and Scientific Text Descriptions. *Journal of Machine Learning*, 2025

Yuxuan Liu, Jingmin Sun, and Hayden Schaeffer. BCAT: A Block Causal Transformer for PDE Foundation Models for Fluid Dynamics. *arXiv preprint arXiv:2501.18972*, 2025

Jingmin Sun, **Yuxuan Liu**, Zecheng Zhang, and Hayden Schaeffer. Towards a Foundation Model for Partial Differential Equations: Multi-Operator Learning and Extrapolation. *Physical Review E*, 2025

Yuxuan Liu, Jingmin Sun, Xinjie He, Griffin Pinney, Zecheng Zhang, and Hayden Schaeffer. PROSE-FD: A Multimodal PDE Foundation Model for Learning Multiple Operators for Forecasting Fluid Dynamics. *NeurIPS Foundation Models for Science Workshop*, 2024

Haixin Wang, Yadi Cao, Zijie Huang, **Yuxuan Liu**, Peiyan Hu, Xiao Luo, Zezheng Song, Wanjia Zhao, Jilin Liu, Jinan Sun, et al. Recent Advances on Machine Learning for Computational Fluid Dynamics: A Survey. *arXiv preprint arXiv:2408.12171*, 2024

Yuxuan Liu, Zecheng Zhang, and Hayden Schaeffer. PROSE: Predicting Multiple Operators and Symbolic Expressions using Multimodal Transformers. *Neural Networks*, 2024

Yuxuan Liu, Scott G. McCalla, and Hayden Schaeffer. Random Feature Models for Learning Interacting Dynamical Systems. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 2023

CHAPTER 1

Introduction

Fluid dynamics and related physical models are essential for describing a wide range of scientific phenomena, from weather forecasting and climate modeling to plasma dynamics and energy generation. Simulating and predicting these dynamics, typically governed by partial differential equations (PDEs), is traditionally complex and computationally demanding due to highly nonlinear interactions, multiscale behavior, and the chaotic effects common in turbulent flows. The challenge is often amplified in real-world applications where measurements of state variables are limited, noisy, or gathered at irregular intervals.

To address these computational bottlenecks, scientific machine learning has emerged as a promising alternative, leveraging deep neural networks to accelerate PDE solving. Initial machine learning approaches utilized established architectures such as Convolutional Neural Networks (CNNs) [GB22] for regular grids, Graph Neural Networks (GNNs) [SGGP$^+$20, LFBC22,CCLJ23] for unstructured meshes, and spatial transformers [Cao21,DHC$^+$22,JLJ$^+$23] for domain-specific dependencies. However, these discrete, grid-bound methods inherently lack an understanding of the continuous physical processes and struggle to generalize to new parameters or varying resolutions. This limitation drove the development of *operator learning*, where neural networks learn the functional mapping between infinite-dimensional function spaces, such as from initial conditions or boundary conditions to the PDE solutions. Prominent architectures like the Deep Operator Network (DeepONet) [LJP$^+$21] and Fourier Neural Operator (FNO) [LKA$^+$21, KLL$^+$23] have successfully learned general solution operators for individual parametric PDEs. Nonetheless, traditional operator learners remain

limited to specific equations or domains: they lack cross-PDE generalizability and require extensive retraining when applied to new physics, geometries, or even different timestep sizes.

Concurrently, the rapid evolution of artificial intelligence has been revolutionized by *foundation models* [BHA+21]. Large Language Models (LLMs) such as GPT [BMR+20], LLaMA [TMS+23], and Gemma [TRP+24], alongside vision models like Stable Diffusion [RBL+22], have demonstrated unprecedented generalization, zero-shot inference, and in-context learning capabilities by leveraging massive, diverse datasets and transformer-based architectures. By pretraining a single large-scale network to understand broad linguistic or visual properties, these models replace the paradigm of training specialized models from scratch for every narrow task.

However, directly applying the foundation models to scientific computing and operator learning introduces unique hurdles. Scientific data inherently differs from text or images: it is driven by strict physical laws, precise deterministic dynamics, and possesses varying state dimensions and properties. While recent efforts have developed multi-physics models (e.g., FourCastNet [KSH+23], MPP [MRSBP+23], and Aurora [BBL+24]), many of these are heavily customized for atmospheric forecasting or rely heavily on fine-tuning with substantial amounts of data. This thesis aims to address the critical challenge of *scalable multi-operator learning*, wherein a single unified model can simultaneously predict heterogeneous spatiotemporal systems across mixed families of PDEs. Solving this challenge requires new network architectures that can effectively process physical context, including symbolic equations or contextual information, to enable zero-shot and few-shot generalizations.

To advance scalable, multi-operator foundation models for fluid dynamics and other complex PDE systems, this thesis proposes three distinct but complementary approaches, each offering unique pathways to encode physical information, process spatiotemporal data, and generalize across diverse datasets.

Chapter 2 presents *PROSE-FD*, a multimodal PDE foundation model that fuses symbolic mathematical descriptions of the governing equations directly with input trajectory

data. By incorporating these symbolic representations alongside observations, PROSE-FD learns a general, non-autoregressive mapping that adapts zero-shot to various fluid dynamics scenarios, including the Navier-Stokes and shallow water equations across different geometries and parameter regimes.

Chapter 3 shifts the focus to few-shot generalization via *Vision In-Context Operator Networks* (VICON). While existing in-context models process spatial points individually (limiting their use in higher dimensions), VICON employs patch-wise vision transformers to efficiently process dense 2D physical fields. VICON learns implicit PDE operator mappings entirely from a few input-output function pairs provided in the context sequence. This approach natively handles arbitrary timestep strides and non-sequential measurements, proving highly robust to realistic regimes such as incomplete or dropped measurement frames.

Chapter 4 introduces the *Block Causal Attention Transformer* (BCAT) for autoregressive rollout generation. Rather than mapping single fixed-window snapshots or individual tokens to future states, BCAT frames spatiotemporal dynamics as a causal sequence of whole-frame blocks. By enforcing causal constraints at the frame level, BCAT enables enhanced temporal dependencies, robust context lengths, and training parallelization. Additionally, this chapter provides a rigorous theoretical analysis establishing the BCAT architecture as a universal approximator for autonomous dynamical system solution operators, validating its capacity to model spatiotemporal dynamics.

These chapters form a diverse set of methodology for modeling spatiotemporal dynamics. While PROSE-FD explores explicit multimodal conditioning (via symbolic equations), VICON leverages implicit few-shot prompting (in-context pairs), and BCAT captures causality-driven temporal rollouts. Together, they demonstrate multiple viable architectures for constructing scalable PDE foundation models capable of revolutionizing complex scientific computing.

CHAPTER 2

PROSE-FD: A Multimodal PDE Foundation Model for Learning Multiple Operators for Forecasting Fluid Dynamics

2.1 Introduction

Fluid dynamics and related physical models are essential for describing a wide range of scientific phenomena and are employed in various applications, including aerodynamics and aircraft design, weather forecasting, petroleum flow, space plasma dynamics and safety, combustion, and more. Simulating and predicting fluid dynamics is often complex and computationally expensive due to the system's highly nonlinear and multiscale behavior, such as the chaotic effects seen in turbulent flows. This challenge is amplified in real-world applications where measurements of state variables are scarce and noisy, some physical variables are unobserved, and boundary effects are unknown. Consequently, a key task in scientific machine learning is to develop models capable of learning general solution operators for fluid dynamics that can handle these issues.

While the success of foundation models in text-based tasks is well-established [RNS+18, TMS+23, TRP+24], their ability to reason and accurately represent quantities in scientific computing (SC) is limited. This is primarily due to the nature of the data, i.e. scientific data has lower information density than text, and due to the high precision requirements in SC problems. Specifically, there is a demand for a level of accuracy in scientific predictions

that general-purpose foundation models are currently not designed to handle effectively.

PDE foundation models have recently been developed to address these limitations. PROSE is a multimodal PDE foundation model that simultaneously learns to predict the values of state variables and derives a symbolic formulation of the governing equations describing the physical system [Sch17, SM17, SZS20]. PROSE has been applied to ordinary differential equations with chaotic behavior [LZS24] and one-dimensional time-dependent nonlinear PDEs [SLZS25], and can incorporate robust fine-tuning and meta-learning strategies [SZS24]. In [SLZS25, SZS24], it was shown that PROSE can generalize physical features to unseen conservation laws, though its capabilities in higher-dimensional PDE systems remain open. ICON employs in-context learning to guide the model in predicting state variables based on examples, which has been shown to generalize predictions for one-dimensional conservation laws. MPP pretrains an autoregressive vision transformer [DBK$^+$21, ELLZ22] to map observations to future states, although the formulation can be unstable for long prediction windows. For weather prediction, FourCastNet uses the Adaptive Fourier Neural Operator model [GML$^+$21], while Aurora utilizes the 3D Swin Transformer [LLC$^+$21] to generate higher-resolution predictions. However, both models specialize in atmospheric forecasting, may lose some high-frequency information, and are tailored to specific problems. Another Swin-Transformer-based model is Poseidon [HRR$^+$24], where the model needs to be finetuned for downstream tasks.

Main Contributions: We present *PROSE-FD*, a pretrained PDE foundational model that uses a new transformer-based deep neural network that leverages state-variable observations and symbolic information to perform operator-based data prediction for fluid dynamics (FD). The model's formulation allows for the inclusion of various sources of information, including mathematical equations that describe the governing physics, in the inputs and/or outputs. Our contributions are listed below.

- We develop a multimodal PDE foundation model for predicting solution operators for shallow water equations and the Navier-Stokes equations with incompressible and

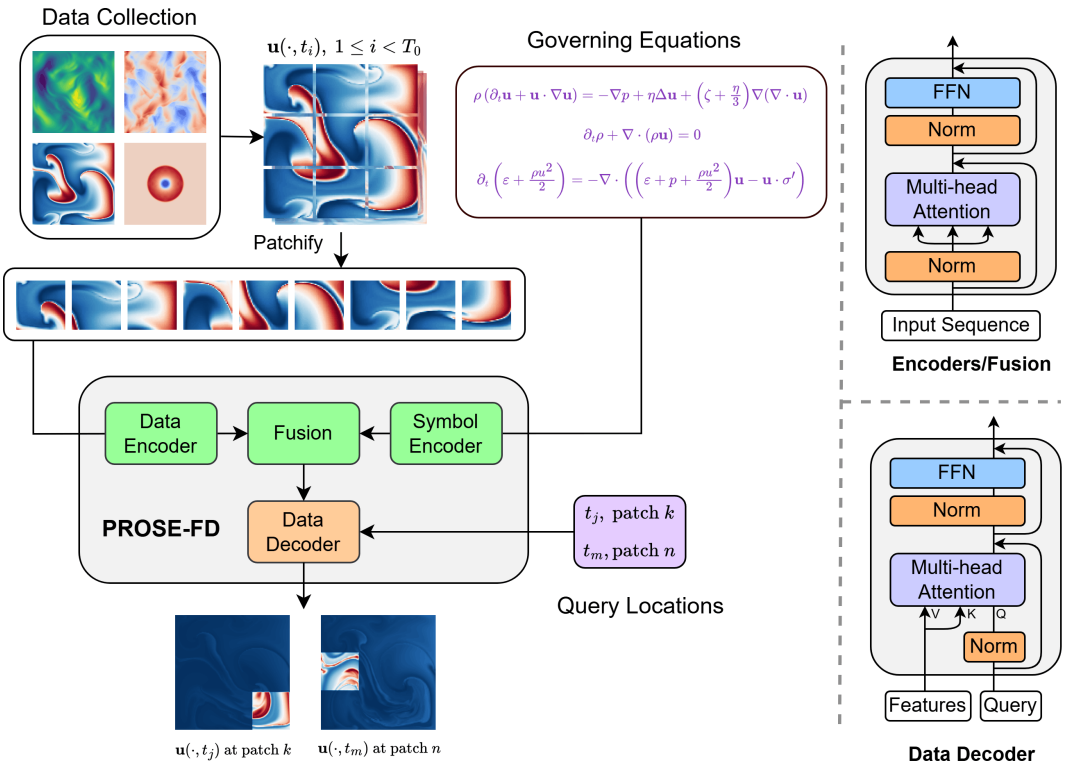


Figure 2.1: **PROSE-FD model overview.** The inputs to the model are the trajectories and governing equations (in symbolic form) sampled from the datasets. The input data is patchified before being converted into a sequence of features, which are then processed with encoders and fused with processed symbolic information. The data decoder takes in fused features and generates predictions at given query locations.

compressible flow, regular and complex geometries, and different buoyancy settings.

- We show that the PROSE-FD model is capable of accurate predictions for fluid dynamics with a range of physical behavior.
- Using 13 datasets, we demonstrate that PROSE-FD is able to outperform single-operator learning approaches, computer vision models, and other multi-physics models, in the prediction and forecasting tasks with gains ranging from 1.2x to 10.1x.

2.2 Related Works

2.2.1 Operator Learning

Operator learning methods are popular approaches to train neural networks as surrogate models for solutions of partial differential equations (PDEs). These methods aim to train deep neural networks (DNNs) to approximate the map from input functions, such as boundary data and initial states, to the solution of the physical system. One advantage of neural operators is their potential for improved cost efficiency during inference [KSH+23, SZS24]. Recent advancements in operator learning include the Deep Operator Network (DeepONet) [LJK19, LJP+21, LMZ23] and the Fourier Neural Operator (FNO) [LKA+21], which have demonstrated promise in scientific applications such as fluid dynamics [LHLA23] and weather prediction [KSH+23]. These methods have been further developed to incorporate various enhancements, such as different input/output meshes [ZLS23], complex geometries [LKC+24], equation information [WWP21], multiscale features [ZML+24a, WLA+22], and distributed learning frameworks [ZML+24b]. However, these approaches are typically designed to learn a single operator and retrained is needed for a different type of PDE, although fine-tuning can be used to handle different equations [SHK+24, ZML+24c].

2.2.2 PDE Foundation Models

A key challenge with operator learning methods is that they are designed to train a single-operator network for one physical system at a time. Since training a deep neural network requires a large amount of data, this process often necessitates costly simulations and experiments. Moreover, because the operator network is trained on a single physical system, the resulting models do not exhibit emergent generalizations of physical properties [SLZS25].

Foundation models in natural language processing and computer vision are deep learning models trained for multiple tasks using large datasets sampled from heterogeneous sources

[BHA+21]. Approaches such as BERT [DCLT18], GPT [RNS+18,RWC+19,BMR+20], DALL-E [RPG+21, RDN+22], Stable Diffusion [RBL+22], and LLAMA [TLI+23, TMS+23], as well as Claude, have demonstrated success in data processing and generative tasks, showing evidence of generalization to new downstream tasks. However, these models have not been directly applicable to scientific computing problems, such as solving forward and inverse problems in PDEs, which require a higher degree of accuracy.

PDE foundation models aim to approximate solution operators for large classes of PDEs within a single DNN. The goal is to learn a general operator that can represent and infer the forward dynamics of distinct physical systems within one model. Thus, the fundamental task is to develop and train a DNN to accurately generalize to unseen physical dynamics that may share features with the trained dynamics. Current PDE foundation models include Predicting Operators and Symbolic Expressions (PROSE) [LZS24, SLZS25, SZS24], In-Context Operator Network (ICON) [YLMO23, YMLO23, YO24], Multiple Physics Pretraining (MPP) [MRSBP+23], Poseidon [HRR+24], DPOT [HSL+24], Fourier Forecasting Network (FourCastNet) [KSH+23], and Aurora [BBL+24].

2.2.3 Multimodal Machine Learning

Multimodal machine learning focuses on building neural networks with the ability to comprehend, reason, and learn from diverse data sources and structures [LBPL19, SMV+19, TB19, LYRK21,XZC23,LZM22]. One major challenge in multimodal machine learning is the fusion and cross-modal interaction of data from heterogeneous sources. As an example, for reasoning in image caption generation, a multimodal model learns both visual features from the image and additional information from the corresponding textual data [XBK+15a, NHK17]. Another example is AI robotics, where the robots need to process information from multimodal sensors, such as cameras, ultrasounds, and radar systems, to make decisions and interact with environments [EKBT21].

2.3 Methods

The main components of PROSE-FD include patch-based data encoding and decoding, symbolic equation encoding, and multimodal information fusion. We provide the problem description and explain the key components of PROSE-FD.

2.3.1 Problem Setting

Consider parametric families of two dimensional time-dependent nonlinear PDEs, whose state-variables of interest are represented by $\boldsymbol{u}(\boldsymbol{x}, t) \in \mathbb{R}^d$ (for some d up to 4 in our tests) with $\boldsymbol{x} \in \Omega \subseteq \mathbb{R}^2$. Given data up to T_0 timestamps, i.e. the sequence:

$$\{\boldsymbol{u}(\cdot, t_i) \mid 0 \leq i < T_0\},$$

the goal of the forward problem is to predict the subsequent T timestamps

$$\{\boldsymbol{u}(\cdot, t_i) \mid T_0 \leq i < T_0 + T\}.$$

More generally, for operator learning, the goal is to learn the solution as a map $(\boldsymbol{x}, t) \mapsto \boldsymbol{u}(\boldsymbol{x}, t)$ for $(\boldsymbol{x}, t) \in \Omega \times [T_0\Delta t, (T_0 + T)\Delta t]$ where $\Delta t = t_{i+1} - t_i$ is the timestep. In our experiments, we set $T_0 = T = 10$, that is, 10 timestamps are given as inputs and we predict, as operator evaluations, 10 future timestamps.

2.3.2 Model Overview

We use transformer layers as the backbone of the PROSE-FD model, similar to PROSE models introduced in [LZS24,SLZS25]. In Figure 2.1, we provide an illustration of our model, where the structure and the components in the gray box are similar to that of PROSE. The input data $\{\boldsymbol{u}(\cdot, t_i) \mid 0 \leq i < T_0\}$ is first converted into patches before being mapped into a sequence of tokens, which is then processed with the Data Encoder consisting of transformer layers. The input equation symbols are tokenized into a sequence of word embeddings, which

are then processed by the Symbol Encoder and fused with data input in Fusion. The Data Decoder takes queries (based on t and the patch) along with the fused features to generate the solution at the specified locations. Notably, the model's computational complexity is linear with respect to the number of query locations, as each query is evaluated independently.

2.3.3 Patch-based Data Encoder

Processing the input data using transformers requires first converting the input into a sequence of tokens. For ODE systems or 1D PDE, directly projecting each timestamp into a token is an effective approach [LZS24, SLZS25]. However, for 2D PDE, a similar approach will lead to information loss due to the curse-of-dimensionality: for space resolution 128×128 and 3 channels (e.g. velocities and pressure), each token needs to encode information of dimension $128^2 \cdot 3 \approx 50K$, which is much larger than the hidden dimension. Patch-based encoding strategies were used in ViT [DBK+21] to balance the sequence length and token dimension. More precisely, each image (i.e. each timestamp $\boldsymbol{u}(\cdot, t)$) is first converted into p^2 patches each of resolution $(128/p) \times (128/p)$. These patches are then transformed and flattened into a sequence of p^2 tokens. We use $p = 8$, thus each token encodes spatial information of dimension $16^2 \cdot 3 = 768$, which is smaller than 1024, the hidden dimension of the linear and attention layers. Consequently, spatially dependent information is retained at the cost of a longer sequence.

We choose $p = 8$ for our inputs, thus the data input is first converted into a sequence of $64 \cdot T_0$ tokens. After adding learnable timestamps and patch positional encodings, the sequence is further processed using self-attention layers. Compared to other video transformer models such as Axial attention [HKWS19], this approach makes it easier to fuse information from different modalities later.

2.3.4 Equation Encoding and Fusion

An important aspect of the simultaneous learning of multi-operator solution operators is to encode the equations so that commonalities and differences can be automatically detected and processed. We encode the equations as symbolic trees [LZS24, SLZS25] (operations and functions as nodes, variables and constants as leaves), which are then converted to a sequence in Polish notation. The sequence is then processed similarly to sentences: they are converted to trainable tokens and then further processed using self-attention. The tokens retain meaning, i.e. mathematical functions such as "tan" and "add" are not further tokenized. Compared to LaTeX encoding of the equation, this approach has a shorter sequence length and easier syntax. For more details, we refer to [JWY23, dKLC22, KdLC22].

PROSE-FD addresses the key challenges in multimodal machine learning by utilizing a fusion layer with self-attention sub-layers [VSP+17, BCB14, XBK+15b] to facilitate effective information exchange between different modalities, achieving a holistic representation. To fuse information obtained from two input modalities, the processed data and symbol sequence are concatenated into a single sequence and further processed through self-attention in the Feature Fusion block. By attending to the symbols, the data sequence obtains information from the symbolic input (e.g., aspects of the equation underlying the data).

2.3.5 Patch-based Operator Decoder

For operator learning, the usual approach is to construct the solution as the map $(\boldsymbol{x}, t) \mapsto \boldsymbol{u}(\boldsymbol{x}, t)$. The PROSE-FD model constructs the solution via cross-attention, where the points $(\boldsymbol{x}, t)$ serve as queries, and the encoded features serve as keys and values. This allows the model to focus on the representations from the fusion stage and specific query locations, thereby strengthening inter-modality relationships. As a result of the curse-of-dimensionality issue, constructing the solution for all $(\boldsymbol{x}, t)$ with cross-attention is computationally expensive due to the increase in sequence length, i.e. the number of spatial points became the num-

Table 2.1: **Main Results and Comparisons with Baselines**. The numbers reported are relative L^2 errors (%). The averages are taken with respect to the 6 distinct families listed in the columns of the table. For each family of equations, we **bold** the best results. *Note that the PDEBench CNS contains 8 subsets of parameter configurations.

Model	Param	PDEBench			PDEArena		CFDBench	Average
		SWE	CNS*	INS	NS	NS-cond	-	
DeepONet	3.5M	3.55	7.41	64.61	35.33	51.85	12.50	29.21
FNO	0.6M	3.71	6.31	36.83	38.68	55.63	8.53	24.95
UNet	5.6M	0.33	3.19	3.43	12.56	16.82	0.76	6.18
ViT	154M	0.30	1.99	3.14	10.19	15.71	0.70	5.34
MPP-B	116M	1.02	1.90	7.52	5.71	12.56	1.23	4.99
MPP-L	407M	0.47	1.53	6.42	4.64	9.64	0.73	3.91
DPOT-M	122M	0.54	**1.01**	5.20	4.92	**8.55**	0.64	3.47
PROSE-FD	165M	**0.14**	1.18	**1.59**	**4.20**	9.76	**0.43**	**2.88**

ber of elements in the sequence. To ensure the independence of the query evaluation while maintaining reasonable computational complexity, instead of constructing a function evaluated for each input spatial location $\boldsymbol{x}$, the Data Decoder block learns a function that maps patches P. That is, given a patch P representing a set of spatial coordinates $P = \{\boldsymbol{x}_k\}_{k \in K}$ (where K is an index set with cardinality of 64 in our applications), the Data Decoder learn the solution

$$(P, t) \mapsto \{\boldsymbol{u}(\boldsymbol{x}_k, t_i) \mid \boldsymbol{x}_k \in P\}.$$

Additionally, due to the linear complexity of the Data Decoder, the output sequence length can be larger than the encoder's sequence length. Consequently, we use $p = 16$ output patches in each dimension.

Table 2.2: **Compare PROSE-FD trained with AdamW and Muon optimizer.** The numbers reported are relative L^2 errors (%). The averages are taken with respect to the 6 families.

Optimizer	Param	PDEBench			PDEArena		CFDBench	Average
		SWE	CNS*	INS	NS	NS-cond	-	
AdamW	165M	0.28	1.41	2.75	5.27	9.61	0.61	3.32
Muon	165M	0.14	1.18	1.59	4.20	9.76	0.43	2.88

2.4 Experiments

In this section, we first explain the experiment setup. We then present the main results and compare our PROSE-FD model with other baseline models, in terms of prediction accuracy and resource usage. Finally, we validate our key architecture choices with ablation studies. More experiment details can be found in Appendix 2.B.

2.4.1 Experiment Setup

Dataset. The dataset we use contains 6 parametric families of PDEs modeling fluid dynamics in different regimes collected from 3 heterogeneous sources: PDEBench [TPL+22], PDEArena [GB22], and CFDBench [LCZ23]. The dataset includes shallow water equations and the Navier-Stokes system with incompressible and compressible flow, regular and complex geometries, and different buoyancy settings. For datasets that do not provide a train/val/test splitting, we use the standard 80%/10%/10% splitting. For more details, we refer to Appendix 2.A.

Evaluation Metric. The relative L^2 norm is used as the evaluation metric. More precisely, given the model's prediction $\tilde{\boldsymbol{u}}$ and the ground truth $\boldsymbol{u}$, we compute the (time-averaged)

relative L^2 error:

$$\frac{1}{T}\sum_{i=T_0}^{T_0+T-1}\frac{\|\boldsymbol{u}(\cdot,t_i)-\tilde{\boldsymbol{u}}(\cdot,t_i)\|_2}{\|\boldsymbol{u}(\cdot,t_i)\|_2+\varepsilon}, \tag{2.1}$$

where $T_0 = 10$ is the number of input steps, $T = 10$ is the number of output steps, and $\varepsilon = 10^{-7}$. Note that for the Navier-Stokes dataset from PDEArena, the temporal grid resolution is only 14, thus we set $T = 4$ for this dataset only. The average used in the last column of Table 2.1 is the average of the relative L^2 errors over the 6 families, i.e. the average over each row of the table.

2.4.2 Baselines and Comparisons

We compare our PROSE-FD model with the following baselines. DeepONet [LJK19] and FNO [LKA+21] are popular single-operator learning methods that efficiently approximate PDE solution operators. UNet [RFB15] is a classical convolution-based image processing model, utilizing symmetric hierarchical structures to capture both context and fine details for pixel-wise predictions. ViT [DBK+21] is a popular transformer-based image processing model that captures global image dependencies and demonstrates scalability for model sizes in image and video processing tasks. Both MPP [MRSBP+23] and DPOT [HSL+24] generate PDE solutions autoregressively, with MPP based on Axial-ViT and DPOT based on Adaptive Fourier Neural Operators. More details about the baselines are included in Appendix 2.B.3.

2.4.3 Main Results

The main experiment results are included in Table 2.1, where we report the relative L^2 error (%) for each family of equations and the average. For all the models, we use the same training setting: a single model is trained to predict all families of equations, without any fine-tuning. The PROSE-FD model achieves the best results in terms of average across all families, outperforming other models with various sizes, including MPP-L which are ∼3x larger. For specific families, the PROSE-FD model achieves the best results for four out of

Table 2.3: **Results for PROSE-FD Ablation Studies.** We compare the baseline (operator) approach to a rollout (recursive) prediction variant and a data-only (no symbolic information) variant. The numbers reported are relative L^2 errors (%). The averages are taken with respect to the 6 distinct families listed in the columns of the table.

Model	Param	PDEBench			PDEArena		CFDBench	Average
		SWE	CNS	INS	NS	NS-cond	-	
Baseline	165M	0.28	1.41	2.75	5.27	9.61	0.61	3.32
Rollout in time	165M	0.23	1.16	4.56	4.48	8.86	1.08	3.39
Data-only	164M	0.29	1.54	2.49	6.43	10.39	0.54	3.61

six families. In Figure 2.2, we provide example output visualizations for PROSE-FD. More visualizations of the PROSE-FD model output and comparisons with other models can be found in Appendix 2.B.4.

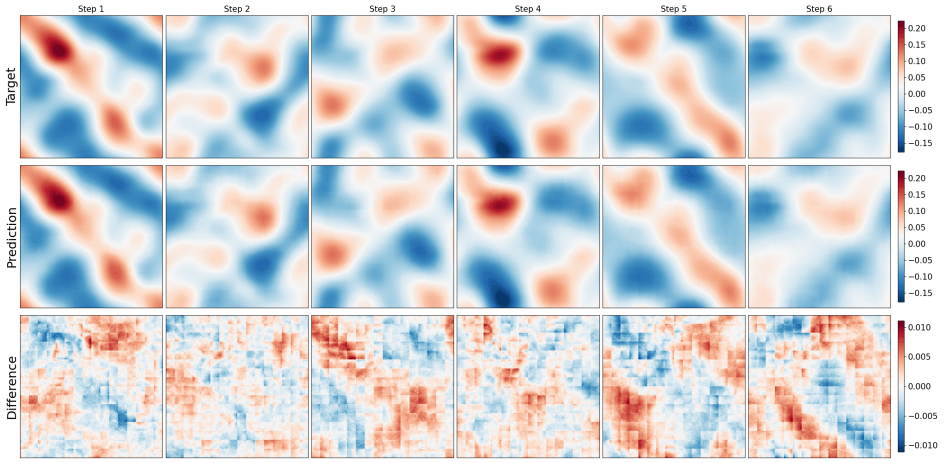
2.4.4 Ablation Studies

In this section, we present the results of our ablation studies to validate some key architecture and optimizer choices. We compare the zero-shot testing performance (relative L^2 error) averaged over the 6 datasets. The results are shown in Table 2.3.

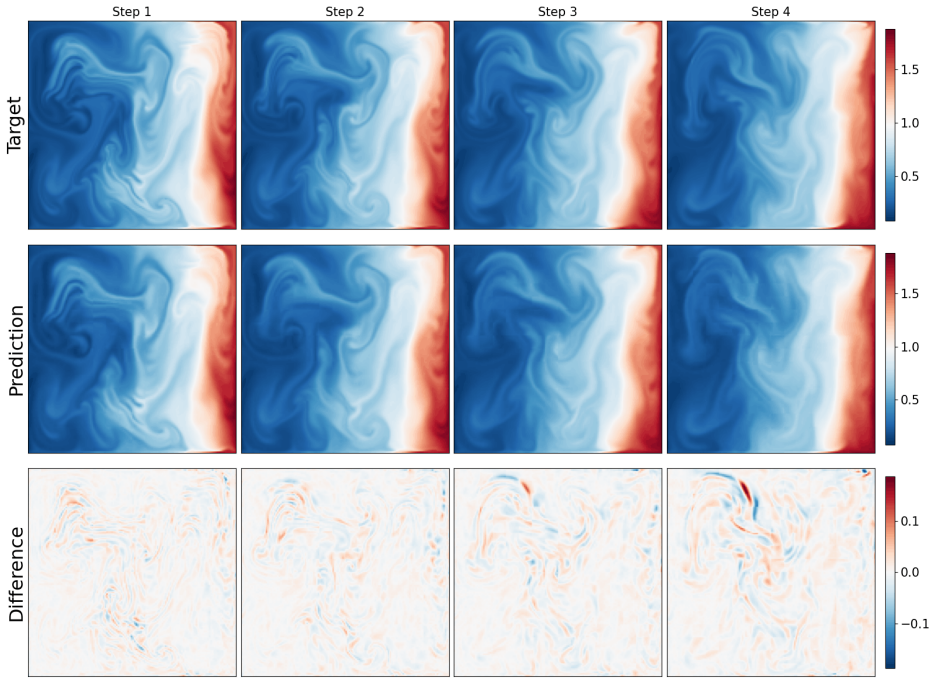
Rollout vs. Operator Formulation. Our baseline PROSE-FD model learns the map from T_0 input timestamps to T output steps in a non-autoregressive way, i.e., it can output multiple future timestamps in a single forward pass. The Data Decoder is an operator in time and no explicit rollout is needed. An alternative strategy is to have the model learn the map from T_0 input timestamps to the next output step, then rollout, i.e. recursively apply, the model during the inference stage [ELLZ22]. To obtain T output steps, the model needs to be called T times to generate the full solution. As shown in Table 2.3, our baseline model outperforms the rollout model, demonstrating that the operator learning approaches

Table 2.4: **Comparing the inference speed and GPU memory usage for all models.** We measure the recourse usage for different models and variants to generate 10 output steps for a single trajectory.

Model	Param	Inference Speed (ms)	GPU Memory Usage (MB)
DeepONet	3.5M	2.7	190
FNO	0.6M	1.7	372
UNet	5.6M	1.3	135
ViT	154M	3.1	2356
MPP-B	116M	207.9	3087
MPP-L	407M	438.3	10869
DPOT-M	122M	95.7	1864
Data-only Variant	164M	7.0	1092
Rollout-in-time Variant	165M	90.4	1025
PROSE-FD	165M	7.5	1269



(a) 6 consecutive output steps for PDEBench Compressible Navier-Stokes dataset. The channel plotted is the y-velocity in equation (2.7). Each column represents a different timestamp. For this trajectory, the relative L^2 error is 0.17%.



(b) 4 consecutive output steps for PDEArena Navier-Stokes dataset. The channel plotted is the particle density field in equation (2.10). Each column represents a different timestamp. For this trajectory, the relative L^2 error is 3.55%.

Figure 2.2: **Two example outputs for the PROSE-FD model.**

can avoid error accumulation. Additionally, the operator formulation is much more efficient than rollout, as shown in Section 2.4.5.

Influence of Symbolic Information. To show the importance of symbolic equation information, we compare our baseline PROSE-FD model with a model that only uses the Data Encoder and the Data Decoder. We increase the number of layers for each component so the final parameter count is close to the baseline model. The comparison in Table 2.3 demonstrates that the symbolic encoding structure of the PROSE-FD model enhances the predictions.

Muon Optimizer vs. AdamW Optimizer. The Adam optimizer [Kin14, LH17] has long been the standard optimizer for training various deep learning models. Recently, the Muon optimizer [JJB+24] was proposed to be a strong alternative to Adam. Empirically, Muon demonstrated strong performance in large language models [LSY+25] and scientific computing [LSS25], with small computational overhead (<1%).

The Muon optimizer is based on SGD with preconditioned momentum, which is designed for optimizing hidden layer 2D parameters (the scalar, vector, input, and output layer parameters are still optimized using AdamW). The preconditioning step orthogonalizes the parameter-update matrices using Newton-Schulz iteration. Observing the recent success of Muon, we train PROSE-FD using a combination of Muon (for transformer 2D parameters) and AdamW (for other parameters). For brevity, in figures and tables, we refer to this as just Muon. We found attention query-key normalization [HDPC20] necessary to stabilize training, which is consistent with [LSY+25].

The results are shown in Table 2.2, where Muon outperforms AdamW in all but one dataset, improving PROSE-FD's performance in most cases. For the dataset where AdamW is slightly better, the performance is very close. In terms of average errors, Muon reduces the error from 3.32% to 2.88%. In Figure 2.3, we visualize and compare the output of PROSE-FD trained using Muon and AdamW. We observe that, when training PROSE-FD with Muon, the error magnitude is lower.

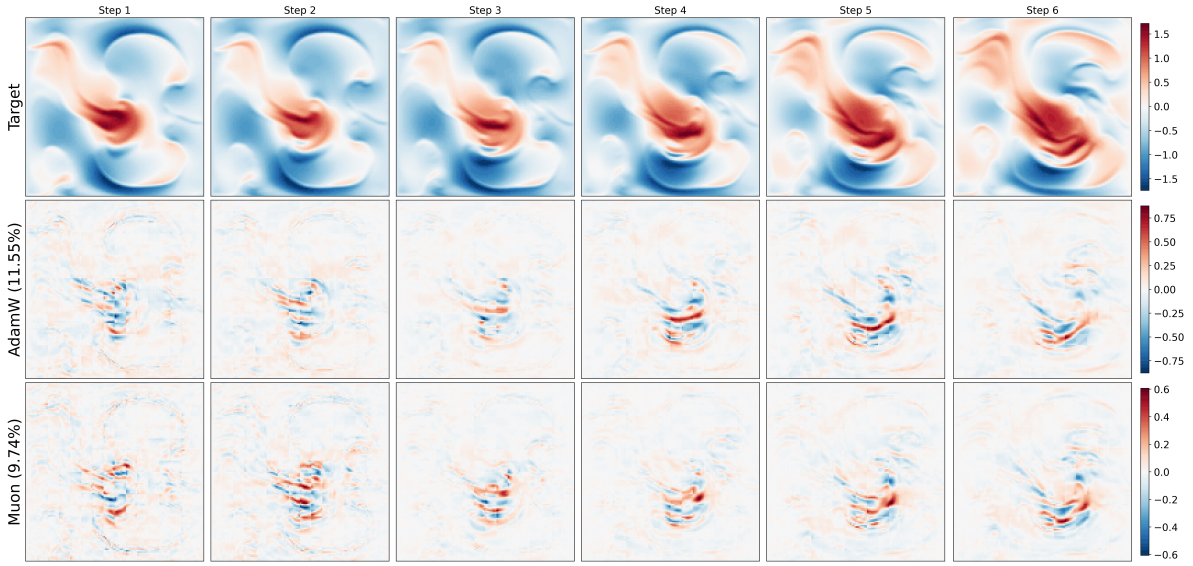


Figure 2.3: **Comparing PROSE-FD trained with Muon vs. AdamW optimizer.** The target (first row) is the first 6 output steps from the PDEArena Navier-Stokes (conditioned) dataset (y-velocity channel). For each optimizer (each row), we display the difference between the target and model output. Relative L^2 errors for the full trajectories are listed after the optimizer names.

2.4.5 Resource Usage Comparison

In Table 2.4, we compare the inference speed and GPU memory usage for all models. To obtain these results, on a single NVIDIA H100 GPU, we use a batch size of 1 and measure the time required to generate 10 output steps for the shallow water equation. To ensure stability, we average the model run time over 50 repeats after some warmup. As PROSE-FD does not require autoregressive generation, the inference speed is faster than models that require rollout (MPP, DPOT, and rollout-in-time variant), demonstrating the efficiency of PROSE-FD's operator framework.

2.5 Conclusion

PROSE-FD is a pretrained PDE foundational model that utilizes transformers to encode and process data and symbolic information for predicting solutions for fluid systems. The multimodal aspect of the approach allows for further experimentation and enhancement by including additional modality information that describes the physical systems of interest. Our model is able to encode information from the two-dimensional shallow water equations and the two-dimensional Navier-Stokes equations with incompressible and compressible flow, regular and complex geometries, and different buoyancy settings into one model. Through extensive testing, we demonstrated that the model's predictions accurately capture the behavior presented in the datasets used – outperforming other single-operator learning and transformer models. Thus, the approach presents a general-purpose surrogate model for two-dimensional fluid systems. Further work will examine the scalability of the model and the encoding of boundary effects.

2.A Dataset Details

The data was obtained from the PDEBench [TPL+22], PDEArena [GB22], and CFDBench [LCZ23] datasets. Unless otherwise specified, the space resolution is 128×128.

2.A.1 PDEBench

Shallow Water Equation. The quantity of interest is the water depth $h(\boldsymbol{x}, t)$ on domain $[-2.5, 2.5]^2 \times [0, 1]$ with Neumann boundary condition. The temporal resolution is 101. The equations are:

$$\partial_t h + \nabla h\boldsymbol{u} = 0, \tag{2.2}$$

$$\partial_t h\boldsymbol{u} + \nabla \left(h\, \boldsymbol{u} \cdot \boldsymbol{u} + \frac{1}{2} g_r h^2 \right) = -g_r h \nabla b. \tag{2.3}$$

Incompressible Navier-Stokes Equation. The quantities of interest are the velocities $\boldsymbol{u}(\boldsymbol{x}, t)$ and particle density $c(\boldsymbol{x}, t)$ on domain $[0, 1]^2 \times [0, 5]$ with Dirichlet boundary condition. The temporal resolution is 1000. The equations are:

$$\rho(\partial_t \boldsymbol{u} + \boldsymbol{u} \cdot \nabla \boldsymbol{u}) = -\nabla p + \mu \Delta \boldsymbol{u} + \mathbf{F}, \tag{2.4}$$

$$\nabla \cdot \boldsymbol{u} = 0, \tag{2.5}$$

$$\partial_t c + \nabla \cdot (c\boldsymbol{u}) = 0. \tag{2.6}$$

The forcing term $\mathbf{F}$ is randomly sampled.

Compressible Navier-Stokes Equation. The quantities of interest are the velocities $\boldsymbol{u}(\boldsymbol{x}, t)$, pressure $p(\boldsymbol{x}, t)$, and density $\rho(\boldsymbol{x}, t)$ on domain $[0, 1]^2 \times [0, 1]$ with periodic boundary conditions. The temporal resolution is 21. For equations with low viscosities, the dataset is provided on a finer 512×512 space grid, which is downsampled to 128×128 for consistency (through average pooling). The equations are:

$$\partial_t \rho + \nabla \cdot (\rho \boldsymbol{u}) = 0, \tag{2.7}$$

$$\rho(\partial_t \boldsymbol{u} + \boldsymbol{u} \cdot \nabla \boldsymbol{u}) = -\nabla p + \eta \Delta \boldsymbol{u} + (\zeta + \eta/3) \nabla (\nabla \cdot \boldsymbol{u}), \tag{2.8}$$

$$\partial_t \left(\varepsilon + \frac{\rho u^2}{2} \right) = -\nabla \cdot \left(\left(\varepsilon + p + \frac{\rho u^2}{2} \right) \boldsymbol{u} - \boldsymbol{u} \cdot \sigma' \right). \tag{2.9}$$

2.A.2 PDEArena

Incompressible Navier-Stokes Equation. The quantities of interest are the velocities $\boldsymbol{u}(\boldsymbol{x}, t)$ and particle density $c(\boldsymbol{x}, t)$ on domain $[0, 32]^2 \times [18, 102]$ with Dirichlet boundary conditions for velocity and Neumann boundary condition for particle field. The temporal resolution is 14. The equations are:

$$\rho(\partial_t \boldsymbol{u} + \boldsymbol{u} \cdot \nabla \boldsymbol{u}) = -\nabla p + \mu \Delta \boldsymbol{u} + \mathbf{F}, \tag{2.10}$$

$$\nabla \cdot \boldsymbol{u} = 0, \tag{2.11}$$

$$\partial_t c + \nabla \cdot (c\boldsymbol{u}) = 0. \tag{2.12}$$

The forcing term $\mathbf{F}$ takes the form $\mathbf{F} = (0, f)$ with $f = 0.5$.

Incompressible Navier-Stokes Equation (Conditioned). The quantities of interest are the velocities $\boldsymbol{u}(\boldsymbol{x}, t)$ and particle density $c(\boldsymbol{x}, t)$ on domain $[0, 32]^2 \times [18, 102]$ with Dirichlet boundary conditions for velocity and Neumann boundary condition for particle field. The temporal resolution is 56. The equations are:

$$\rho(\partial_t \boldsymbol{u} + \boldsymbol{u} \cdot \nabla \boldsymbol{u}) = -\nabla p + \mu \Delta \boldsymbol{u} + \mathbf{F}, \tag{2.13}$$

$$\nabla \cdot \boldsymbol{u} = 0, \tag{2.14}$$

$$\partial_t c + \nabla \cdot (c\boldsymbol{u}) = 0. \tag{2.15}$$

The forcing term $\mathbf{F}$ takes the form $\mathbf{F} = (0, f)$ where f is uniformly sampled in $[0.2, 0.5]$.

2.A.3 CFDBench

Incompressible Navier-Stokes Equation. The quantities of interest are the velocities $\boldsymbol{u}(\boldsymbol{x}, t)$ and pressure $p(\boldsymbol{x}, t)$. This dataset contains irregular geometries with Dirichlet boundary conditions. The raw space resolution is 64×64 which is upsampled to 128×128 via interpolation. The equations are:

$$\rho(\partial_t \boldsymbol{u} + \boldsymbol{u} \cdot \nabla \boldsymbol{u}) = -\nabla p + \mu \Delta \boldsymbol{u}, \tag{2.16}$$

$$\nabla \cdot \boldsymbol{u} = 0. \tag{2.17}$$

2.B Experiment Details

We provide more details about the training process, architecture, and baselines.

2.B.1 Training

We perform data normalization during the training process. Given the input sequence of data $\{\boldsymbol{u}(\cdot, t_i) \mid 0 \leq i < T_0\}$, we compute the mean and standard deviation of each input

trajectory, which are used to normalize both the input and ground truth sequence. The loss function is the standard mean squared error in the normalized space. The models are trained using a combination of Muon and AdamW optimizer with a global batch size of 160 for 40 epochs where each epoch is 4,000 steps. The warmup-stable-decay learning rate scheduler [HTH+24] is used with 10% warmup and 50% decay. We use learning rate 7×10^{-4} and weight decay 10^{-4}. On four NVIDIA H100 GPUs, the training takes about 24 hours.

2.B.2 Model Hyperparameters

The model hyperparameters are summarized in Table 2.5.

2.B.3 Baselines

In this section, we include more details about the compared models.

DeepONet [LJK19]. We employ the unstacked DeepONet architecture, consisting of a single trunk network and a single branch network. Initially, the input data is divided into 8×8 patches, with each patch being embedded into a 128-dimensional vector. These vectors are then passed through the branch network, producing an output with a basis dimension of $p = 50$. Simultaneously, the query point is processed through the trunk network, which also outputs a vector with the same dimension, p. The output solution at the query point is obtained by taking the inner product of the outputs from the two networks.

Table 2.5: **Model hyperparameters.** FFN means feedforward network.

Hidden dimension - attention	1024	Hidden dimension - FFN	2048
Number of attention heads	8	Fusion attention layers	10
Data encoder attention layers	4	Data decoder attention layers	4
Symbol encoder attention layers	1	Dropout	0
Input patch number	8	Output patch number	16
PreNorm	RMS Norm		

FNO [LKA+21]. We use 4 layers of standard 3d FNO to process the input data. The number of modes to keep in each dimension is set to 8, and the number of hidden channels is set to 16. The 3d FNO model directly maps 10 input steps to 10 output steps.

UNet [RFB15]. We use 8 layers of 3d UNet with GeLU activation and 32 hidden dimensions. The 3d UNet model directly maps 10 input steps to 10 output steps (Padding is added to the input to make it a power of 2, which is needed for the shrinkage in dimension).

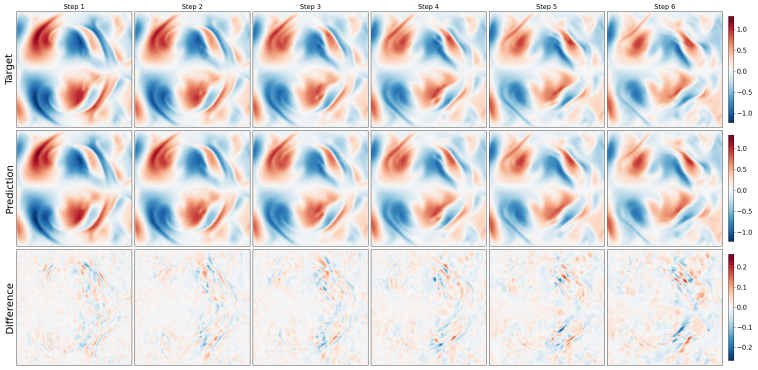
ViT [DBK+21]. For ViT, we use 10 transformer encoder layers similar to the one shown in Figure 2.1. The input patch number is set to be 8, the hidden dimension for attention is 1024, the hidden dimension for the feedforward network is 2048, and the number of heads is 8. The ViT model directly maps 10 input steps to 10 output steps.

MPP [MRSBP+23]. In [MRSBP+23], MPP is trained on a different training dataset and different evaluation metrics are used. For a fair comparison, we retrained MPP-B and MPP-L on our training dataset keeping the same model configurations and optimizer hyperparameters. We evaluate MPP-B and MPP-L using the same testing setup and metric, where rollout is used to obtain all 10 output steps.

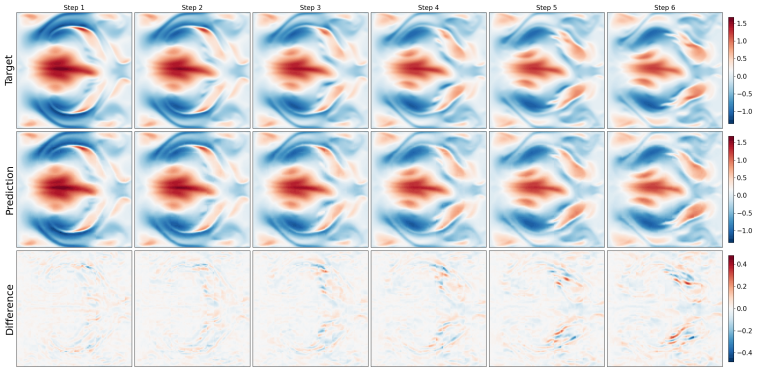
DPOT [HSL+24]. In [HSL+24], DPOT is trained on a different training dataset and different evaluation metrics are used. For a fair comparison, we retrained DPOT-M on our training dataset keeping the same model configurations and optimizer hyperparameters. We evaluate DPOT-M using the same testing setup and metric, where rollout is used to obtain all 10 output steps.

2.B.4 More Visualizations

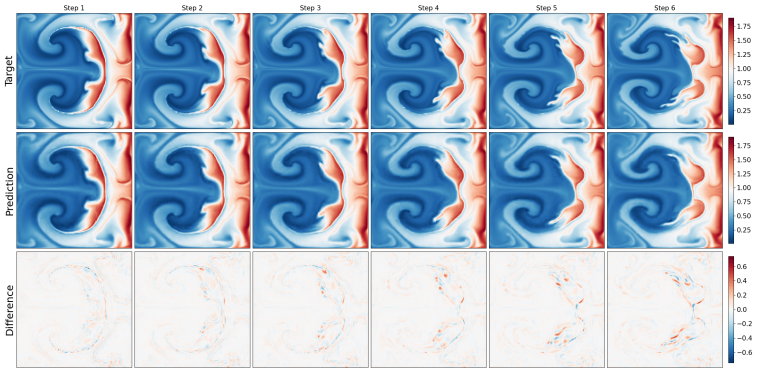
See Figure 2.4 for PROSE-FD model output visualizations, and Figure 2.5 for additional comparison between outputs from different models.



(a) The channel plotted is the x-velocity.



(b) The channel plotted is the y-velocity.



(c) The channel plotted is the particle density.

Figure 2.4: **Example outputs for the PROSE-FD model.** 6 consecutive output steps for PDEArena NS-cond dataset (all three channels in equation (2.13)). Each column represents a different timestep. For this trajectory, the relative L^2 error is 7.79%. This example shows that even when the relative error is higher, the structures of the flow can still be predicted correctly.

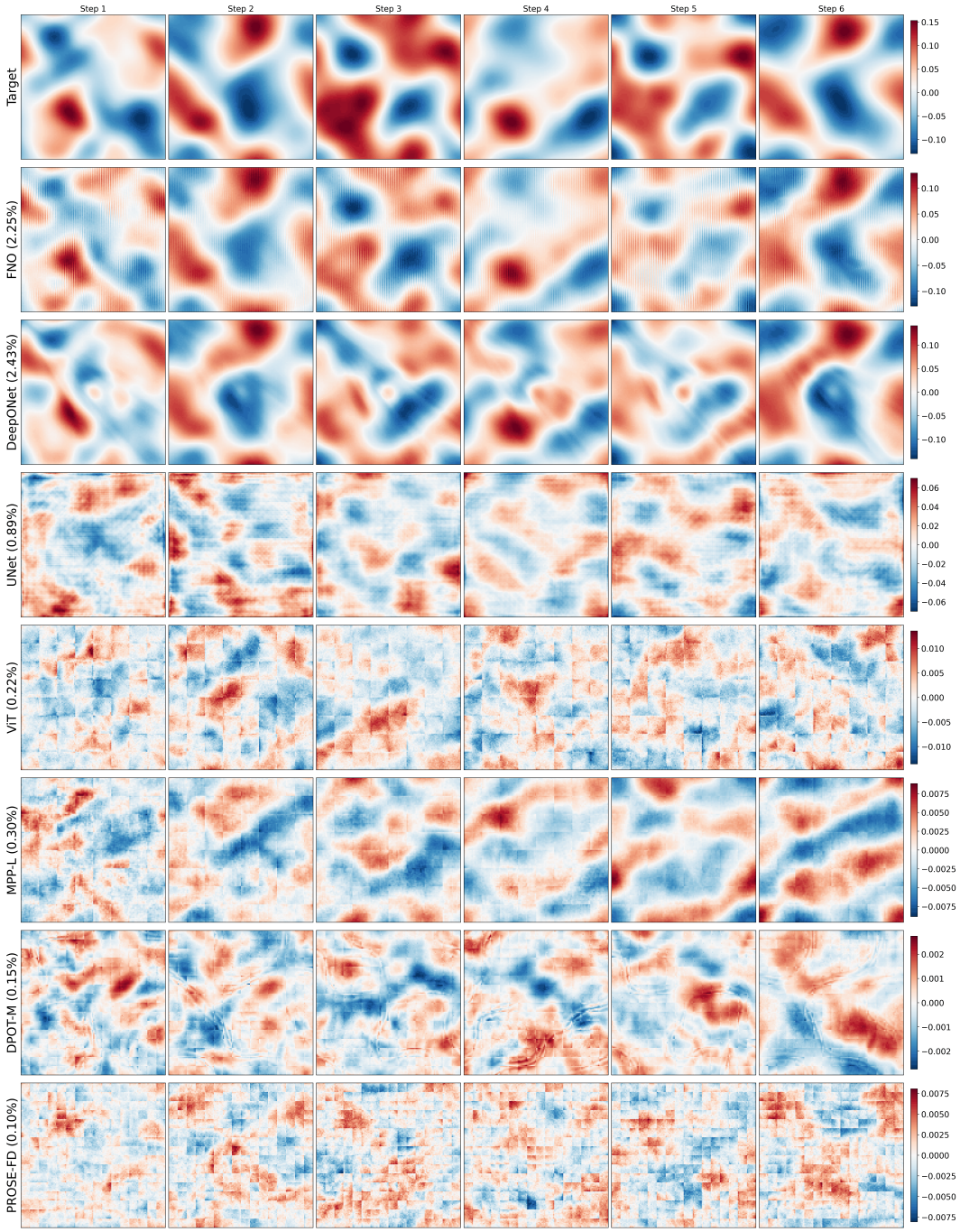


Figure 2.5: **Comparing outputs from different models.** The target (first row) is the first 6 output steps from the PDEBench Compressible Navier-Stokes dataset (x-velocity). For each model (each row), we display the difference between the target and model output. Relative L^2 errors for the full trajectories are listed after the model names.

CHAPTER 3

VICON: Vision In-Context Operator Networks for Multi-Physics Fluid Dynamics Prediction

3.1 Introduction

Machine learning has emerged as a powerful tool for solving Partial Differential Equations (PDEs). Traditional approaches primarily operate on discrete representations, with Convolutional Neural Networks (CNNs) excelling at processing regular grids [GB22], Graph Neural Networks (GNNs) handling unstructured meshes [SGGP+20, LFBC22, CCLJ23], and transformers capturing dependencies between sampling points within the same domains [Cao21, DHC+22, JLJ+23]. However, these discrete approaches face a fundamental limitation: they lack PDE context information (such as governing parameters) and consequently cannot generalize to new parameters or PDEs beyond their training set.

This limitation led to the development of "operator learning," where neural networks learn mappings from input functions to output functions, such as from initial/boundary conditions to PDE solutions. This capability was first demonstrated using shallow neural networks [CC95b, CC95a], followed by specialized architectures including the widely adopted Deep Operator Network (DeepONet) [LJP+21], Fourier Neural Operator (FNO) [LKA+21, KLL+23], and recent transformer-based operator learners [LMF22, LSBF23, OKS+24]. While these methods excel at learning individual parametric PDEs, they cannot generalize across different PDE types, necessitating costly retraining for new equations or even different timestep sizes.

Inspired by the success of large language models in multi-domain generation, researchers have explored "multi-physics PDE models" to address this single-operator limitation. Common approaches employ pretraining and fine-tuning strategies [CSR+24,HRR+24,ZML+24c], though these require substantial additional data (typically hundreds of frames) and fine-tuning before deployment. This requirement severely limits their practical applications in scenarios like online control and data assimilation, where rapid adaptation with limited data sampling is essential.

To eliminate the need for additional data collection and fine-tuning, several works have attempted to achieve few-shot or zero-shot generalization across multi-physics systems by incorporating explicit PDE information. For example, the PROSE approach [LZS24, LSH+24, SLZS25, JSZS24] processes the symbolic form of governing equations using an additional transformer branch. Similarly, PDEformer [YHC+24] converts PDEs into directed graphs as input for graph transformers, while others embed symbol-tokens directly into the model [LLF24]. Despite these advances, such methods require explicit knowledge of the underlying PDEs, which is often unavailable when deploying models in new environments.

To further lower barriers for applying multi-physics models in real-world applications, the In-Context Operator Network (ICON) [YLMO23] offers a fundamentally different approach: ICON implicitly encodes system dynamics through a few input-output function pairs, then extracts dynamics from these pairs in an in-context fashion. This approach enables few-shot generalization without retraining or explicit PDE knowledge, while the minimal required input-output pairs can be readily collected during deployment. A single ICON model has demonstrated success in handling both forward and inverse problems across various ODEs, PDEs, and mean-field control scenarios.

However, ICON's computational efficiency becomes a critical bottleneck when handling dense data in higher dimensions. Specifically, ICON treats each sampled spatial point as an individual token, leading to quadratic computational complexity with respect to the number of points. This makes it computationally prohibitive for practical 2D and 3D appli-

cations—to date, only one instance of a 2D case using sparse data points has been demonstrated [YLMO23].

To address this limitation, we propose *Vision In-Context Operator Networks* (VICON), which leverage vision transformers to process 2D functions using an efficient patch-wise approach. Going beyond classic vision transformers that typically process patches from a single image, VICON extends the architecture to handle sequences of input-output function pairs, enabling "next function prediction" capabilities while preserving the benefits of in-context operator learning. The current implementation focuses on passive, autonomous PDEs; extension to non-autonomous systems with external forcing is left for future work.

Our contributions include:

- Development of VICON, a vision transformer-based in-context operator network for two-dimensional time-dependent PDEs that maintains the flexibility of in-context operator learning while efficiently handling dense data in higher dimensions.
- Comprehensive evaluation across diverse fluid dynamics systems comprising 879K frames with varying timestep sizes, demonstrating substantial improvements over state-of-the-art models MPP [MRSBP$^+$23] and DPOT [HSL$^+$24] in both accuracy and efficiency. Our approach reduces the averaged last-step rollout error by 37.9% compared to DPOT and 44.7% compared to MPP, while requiring only 72.5% and 34.8% of their respective inference times.
- Enhanced flexibility supporting varying timestep strides and non-sequential measurements, offering substantial robustness in realistic scenarios with imperfect data collection, e.g. with missing frames. Under these scenarios, VICON exhibits only 24.41% relative performance degradation versus 71.37%-74.49% degradation in baseline methods.

3.2 Related Work

Operator Learning. Operator learning [CC95b, CC95a, LKA+21, LJP+21] addresses the challenge of approximating operators $G : U \to V$, where U and V are function spaces representing physical systems and differential equations, e.g., G maps initial/boundary conditions or system parameters to corresponding solutions. Among the various operator learning approaches, DeepONets [LJP+21, LMC+22] employ branch and trunk networks to independently process inputs and query points, while FNOs [LKA+21] leverage fast Fourier transforms to efficiently compute kernel integrations for PDE solutions in regular domains. These methods have been extended to incorporate equation information [WWP21, LZK+24], multi-scale features [ZML+24a, WLA+22, LZL+24], and adaptation to heterogeneous and irregular meshes [ZLS23, ZML+24b, LKC+24, WLW+24].

However, these approaches typically learn a single operator, requiring retraining when encountering different PDE types or timestep sizes. Our work aims to develop a unified model for multiple PDEs with few-shot generalization capabilities based on limited observation frames.

Multi-Physics PDE Models. Foundation models in natural language processing [BMR+20, TLI+23] and computer vision [RPG+21] have demonstrated remarkable versatility across diverse tasks. Drawing inspiration from this paradigm, recent research has explored developing unified models for multiple PDEs in domains such as PDE discovery [Sch17] and computational fluid dynamics [WCH+24]. Several approaches follow pretraining and fine-tuning strategies [CSR+24, HRR+24], though these require additional resources for data acquisition and fine-tuning before deployment. In parallel, researchers have developed zero- or few-shot multi-physics models by incorporating additional information; for instance, the PROSE approach [LZS24, LSH+24, SLZS25, JSZS24, SZS24] directly encodes symbolic information of PDEs into the model. Other methods achieving zero- or few-shot generalization include using physics-informed tokens [LLF24], representing PDE structures as graphs [YHC+24],

or conditioning transformers on PDE descriptions [ZMW+25]. However, all these techniques require prior knowledge of the PDEs as additional inputs. We select two state-of-the-art models, MPP [MRSBP+23] and DPOT [HSL+24], as our baselines.

In-Context Operator Networks. ICONs [YLMO23, YMLO23, YO24] learn operators by observing input-output function pairs, enabling few-shot generalization across various PDEs without requiring explicit PDE representations or fine-tuning. These networks have demonstrated success in handling forward and inverse problems for ODEs, PDEs, and mean-field control scenarios. However, ICONs face significant computational challenges when processing dense data, as they represent functions through scattered point tokens, resulting in quadratic computational complexity with respect to the number of sample points. This limitation has largely restricted their application to 1D problems, with only sparse sampling feasible in 2D cases [YLMO23]. Our work addresses these limitations by introducing a vision transformer architecture that efficiently handles dense 2D data while preserving the benefits of in-context operator learning.

3.3 Preliminaries

ICONs were introduced and developed in [YLMO23, YMLO23, YO24]. They adapt the in-context learning framework from large language models, aiming to train a model that can learn operators from prompted function examples.

Denote an ICON model as $\mathcal{T}_\theta$, where $\mathcal{T}$ is a transformer with trainable parameters θ. The model takes input as a sequence comprising I pairs of conditions ($\boldsymbol{c}$) and quantities of interest ($\boldsymbol{q}$) as $\{\langle \boldsymbol{c}_i, \boldsymbol{q}_i \rangle\}_{i=1}^{I}$, where each $\boldsymbol{c}, \boldsymbol{q}$ contains multiple tokens representing a single function. The model outputs tokens representing future functions at the positions of $\boldsymbol{c}$, i.e., "next function prediction" similar to "next token predictions" in LLMs. More precisely, for

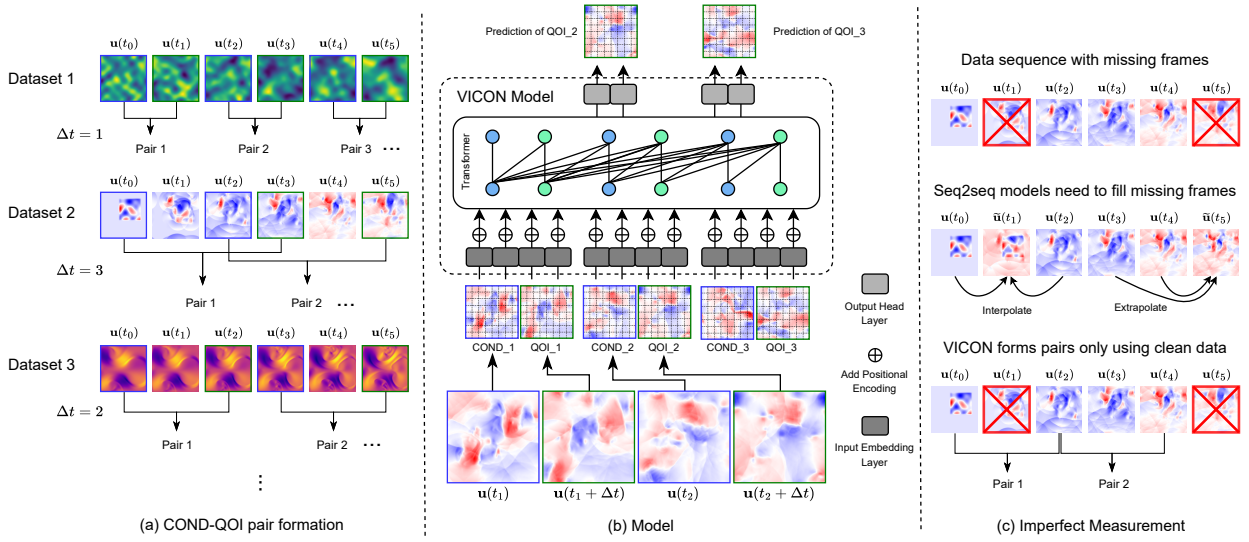


Figure 3.1: **VICON model overview.** (a) The formation process for conditions (COND) and quantities of interest (QOI) pairs. Δt is randomly sampled during training. (b) Model illustration. The inputs to the model are pairs of COND and QOI, which are patchified and flattened before feeding into the transformer layers. The outputs, which represent different patches in the output frame, are transformed back to obtain the final predictions. (c) With imperfect temporal measurements, VICON forms pairs using only clean data, and does not need to fill missing frames.

$J \in \{1, \ldots, I-1\}$, the model predicts $\tilde{\boldsymbol{q}}_{J+1}$ given the leading J pairs and the condition $\boldsymbol{c}_{J+1}$:

$$\tilde{\boldsymbol{q}}_{J+1} = \mathcal{T}_\theta[\boldsymbol{c}_{J+1}; \{\langle \boldsymbol{c}_i, \boldsymbol{q}_i \rangle\}_{i=1}^{J}]. \tag{3.1}$$

For training parallelization, ICON uses a special causal attention mask to perform autoregressive learning (i.e., the model only sees the leading J pairs and $\boldsymbol{c}_{J+1}$ when predicting $\tilde{\boldsymbol{q}}_{J+1}$) and enables output of all predictions within a single forward pass [YMLO23]:

$$\{\tilde{\boldsymbol{q}}_i\}_{i=1}^{I} = \mathcal{T}_\theta[\{\langle \boldsymbol{c}_i, \boldsymbol{q}_i \rangle\}_{i=1}^{I}]. \tag{3.2}$$

Training loss is computed as the mean squared error (MSE) between the predicted states $\tilde{\boldsymbol{q}}_i$ and the ground truth states $\boldsymbol{q}_i$. To ensure that the model receives sufficient contextual

information about the underlying dynamics, we only compute errors for the indices $i > I_{\min}$, effectively requiring at least $I_{\min}$ examples in context. This approach has shown empirical improvements in model performance.

After training, ICON can process a new set of $\langle \boldsymbol{c}, \boldsymbol{q} \rangle$ pairs in the forward pass to learn operators in-context and employ them to predict future functions using new conditions:

$$\tilde{\boldsymbol{q}}_k = \mathcal{T}_\theta[\boldsymbol{c}_k; \{\langle \boldsymbol{c}_i, \boldsymbol{q}_i \rangle\}_{i=1}^{J}], \tag{3.3}$$

where $J \in \{I_{\min}, \ldots, I-1\}$ is the number of in-context examples, $I_{\min}$ is the number of context examples exempted from loss calculation, and $k > I$ is the index of the condition for future prediction.

Importantly, all pairs in the same sequence must be formed with the same operator mapping (i.e., the same PDE and consistent timestep size within one sequence), while operators can vary across different sequences/rows in the training batch to enable the model's generalization ability to different combinations of PDEs and timestep sizes.

3.4 Methodology

3.4.1 Problem Setup

We consider the forward problem for multiple time-dependent PDEs that are *temporally homogeneous Markovian*, defined on domain $\Omega \subseteq \mathbb{R}^2$ with solutions represented by $\mathbf{u}(\boldsymbol{x}, t) : \Omega \times [0, T] \to \mathbb{R}^c$, where c is the number of channels. Given the initial I_0 frames, $\{\mathbf{u}_i = \mathbf{u}(\cdot, t_i) \mid i = 1, \ldots, I_0\}$, the task is to predict future solutions.

Each solution frame is discretized as a three-dimensional tensor $\mathbf{u}_t = \mathbf{u}(\cdot, t) \in \mathbb{R}^{N_x \times N_y \times c}$, where N_x and N_y are the spatial grid sizes. Due to the homogeneous Markovian property, given the initial data $\{\mathbf{u}_t \mid t \geq 0\}$ from some PDE indexed by i_p, for a fixed Δt, there exists an operator $\mathcal{L} = \mathcal{L}_{\Delta t}^{(i_p)}$ that maps frame $\mathbf{u}_t$ to frame $\mathbf{u}_{t+\Delta t}$ for any $t \geq 0$. Our goal is to train a model that learns these operators $\mathcal{L}_{\Delta t}^{(i_p)}$ from the sequence of function pairs generated from

the initial frames. Notably, even within the same dataset and fixed Δt, different trajectories can exhibit different dynamics (e.g., due to different Reynolds numbers Re). These variations are implicitly captured by the function pairs in our framework.

3.4.2 Vision In-Context Operator Networks

The original ICONs represent functions as scattered sample data points, where each data point is projected as a single token. For higher spatial dimensions, this approach necessitates an excessively large number of sample points and extremely long sequences in the transformer. Due to the inherent quadratic complexity of the transformer, this approach becomes computationally infeasible for high-dimensional problems.

Inspired by the Vision Transformer (ViT) [DBK+21], we address these limitations by dividing the physical fields into patches, where each patch is flattened and projected as a token. This approach, which we call Vision In-Context Operator Networks (VICON), addresses the computational limitation of treating each spatial point as an individual token, which leads to quadratic computational complexity with respect to the number of points. This patch-wise approach has its forward process illustrated in Figure 3.1(b) and detailed in the following.

First, the input $\boldsymbol{u}_t$ and output $\boldsymbol{u}_{t+\Delta t}$ functions are divided into patches $\{\boldsymbol{C}^k \in \mathbb{R}^{R_x \times R_y \times c}\}_{k=1}^{N_c}$ and $\{\boldsymbol{Q}^l \in \mathbb{R}^{R_x \times R_y \times c}\}_{l=1}^{N_q}$, where R_x, R_y are the resolution dimensions of the patch and N_c and N_q are the number of patches for the input and output functions, respectively. While N_q can differ from N_c (for instance, when input functions require additional boundary padding, resulting in $N_c > N_q$), we maintain $N_c = N_q$ throughout our experiments. For notational clarity, we add the subscript $i \in \{1, \ldots, I\}$ to denote the pair index, where $\boldsymbol{C}_i^k$ represents the k-th patch of the input function and $\boldsymbol{Q}_i^l$ represents the l-th patch of the output function in the i-th pair. These patches are then projected into a unified d-dimensional latent

embedding space using a shared learnable linear function $f_\phi : R_x \times R_y \times c \to \mathbb{R}^d$:

$$\hat{\boldsymbol{c}}_i^k = f_\phi(\boldsymbol{C}_i^k), \quad \hat{\boldsymbol{q}}_i^l = f_\phi(\boldsymbol{Q}_i^l), \tag{3.4}$$

where $k = 1, \ldots, N_c$ and $l = 1, \ldots, N_q$ are the index of patches. We use the same projection f_ϕ for both conditions and QoIs because in our setup, both represent the same physical fields at different timesteps; sharing the projection enforces consistency in latent space regardless of whether a frame serves as context or target.

We inject two types of learnable positional encoding before feeding the embeddings into the transformer: (1) patch positional encodings to indicate relative patch positions inside the whole domain, denoted as $\mathbf{E}_p \in \mathbb{R}^{N_p \times d}$, where $N_p = \max\{N_c, N_q\}$ and the encoding varies per patch (broadcast across points within each patch); (2) function positional encodings to indicate whether a patch belongs to an input function (using $\mathbf{E}_c \in \mathbb{R}^{I \times d}$) or output function (using $\mathbf{E}_q \in \mathbb{R}^{I \times d}$), as well as their indices in the sequence, where the encoding varies per condition/QoI in each pair (broadcast across patches and points of that condition/QoI):

$$\boldsymbol{c}_i^k = \hat{\boldsymbol{c}}_i^k + \mathbf{E}_p(k) + \mathbf{E}_c(i), \quad \boldsymbol{q}_i^l = \hat{\boldsymbol{q}}_i^l + \mathbf{E}_p(l) + \mathbf{E}_q(i). \tag{3.5}$$

The embeddings $\boldsymbol{c}_i^k$ and $\boldsymbol{q}_i^l$ are then concatenated to form the input sequence for the transformer:

$$\mathbf{c}_1^1 \ldots \mathbf{c}_1^{N_c}, \mathbf{q}_1^1 \ldots \mathbf{q}_1^{N_q}, \ldots, \mathbf{c}_I^1 \ldots \mathbf{c}_I^{N_c}, \mathbf{q}_I^1 \ldots \mathbf{q}_I^{N_q}.$$

To support autoregressive prediction similar to Equation (3.2), VICON employs an alternating-sized (in the case where $N_c \neq N_q$) block causal attention mask, as opposed to the conventional triangular causal attention mask as in mainstream generative large language models [BMR+20] and large vision models [BGM+24]. The mask is defined as follows:

$$\mathbf{M} = \begin{bmatrix} \mathbf{1}_{N_c,N_c} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{1}_{N_q,N_c} & \mathbf{1}_{N_q,N_q} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{1}_{N_c,N_c} & \mathbf{1}_{N_c,N_q} & \cdots & \mathbf{1}_{N_c,N_c} & \mathbf{0} \\ \mathbf{1}_{N_q,N_c} & \mathbf{1}_{N_q,N_q} & \cdots & \mathbf{1}_{N_q,N_c} & \mathbf{1}_{N_q,N_q} \end{bmatrix} \tag{3.6}$$

where $\mathbf{1}_{m\times n}$ denotes an all-ones matrix of dimension $m \times n$, and $\mathbf{0}$ represents a zero matrix of the corresponding dimensions.

After obtaining the output tokens from Equation (3.2), we extract the tokens corresponding to the input patch indices $\mathbf{c}_i^1, \ldots, \mathbf{c}_i^{N_c}$ and denote them as $\tilde{\boldsymbol{q}}_i^1, \ldots, \tilde{\boldsymbol{q}}_i^{N_c}$. These tokens are then projected back to the original physical space using a shared learnable linear function $g_\psi : \mathbb{R}^d \to \mathbb{R}^{R_x \times R_y \times c}$:

$$\tilde{\boldsymbol{Q}}_i^l = g_\psi(\tilde{\boldsymbol{q}}_i^l), \tag{3.7}$$

which predicts $\boldsymbol{Q}_i^l$.

3.4.3 Prompt Normalization

To address the varying scales across different channels and prompts (i.e., sequences of pairs $\{\langle \boldsymbol{c}_i, \boldsymbol{q}_i \rangle\}_{i=1}^I$), we normalize the data before feeding them into the model. A crucial requirement is to consistently normalize functions within the same sequence, to ensure that the same operator is learned. Denoting the normalization operators for $\boldsymbol{c}$ and $\boldsymbol{q}$ as $\mathcal{N}_c$ and $\mathcal{N}_q$ respectively, and the operator in the original space as $\mathcal{L}$, the operator in the normalized space $\mathcal{L}'$ follows:

$$\mathcal{L}'(\mathcal{N}_c(\mathbf{u})) = \mathcal{N}_q(\mathcal{L}(\mathbf{u})). \tag{3.8}$$

In this work, we simply set $\mathcal{N}_c = \mathcal{N}_q$, mainly because our maximum timestep stride $s_{\max} = 5$ is relatively small (i.e., the scale distribution does not change dramatically). Specifically, given input sequence $\{\langle \boldsymbol{c}_i, \boldsymbol{q}_i \rangle\}_{i=1}^I$, we compute the channel-wise mean $\mu =$

$\texttt{mean}(\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_I) \in \mathbb{R}^d$ and standard deviation $\sigma = \texttt{std}(\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_I) \in \mathbb{R}^d$ of all conditions $\{\boldsymbol{c}_i\}_{i=1}^I$, which are then used to normalize both the conditions and QoIs. Training MSE loss is computed in the normalized space. To avoid division by zero, we set a minimum threshold of 10^{-4} for the standard deviation.

3.4.4 Datasets and Data Augmentation

We evaluate on three fluid dynamics datasets representing different physical regimes:

1. PDEArena-Incomp [GB22] (incompressible Navier-Stokes equations), 2,496/608/608 trajectories (train/valid/test) with 56 timesteps each,
2. PDEBench-Comp-HighVis [TPL+22] (compressible Navier-Stokes with high viscosity), containing 40,000 trajectories of 21 timesteps each,
3. PDEBench-Comp-LowVis [TPL+22] (compressible Navier-Stokes with numerically zero viscosity), containing 4,000 trajectories of 21 timesteps each.

For PDEArena-Incomp, we follow the original data split in [GB22]. For PDEBench-Comp-HighVis and PDEBench-Comp-LowVis, we randomly split trajectories in 80%/10%/10% proportions for training/validation/testing, respectively. For all datasets during rollout, we set $I_0 = 10$, i.e., we predict trajectories up to the last timestep given the initial 10 frames.

The temporally homogeneous Markovian property (Section 3.4.1) enables natural data augmentation for training and flexible rollout strategy for inference (Section 3.4.5): by striding with larger timesteps to reduce the number of autoregressive steps in long-term prediction tasks: $\Delta t = s\Delta\tau_{i_p} \mid s = 1, 2, \ldots, s_{\max}$, where τ_{i_p} is the timestep size for recording in trajectory i_p. During training, for each trajectory in the training set, we first sample a stride size $s \sim \mathcal{U}\{1, \ldots, s_{\max}\}$, then form $\langle \mathbf{u}_t, \mathbf{u}_{t+s\Delta\tau} \rangle$ pairs for the corresponding operator. We illustrate this augmentation process in Figure 3.1(a), where different strides are randomly

sampled during dataloading.

Additional dataset details appear in Section 3.A.

3.4.5 Inference with Flexible Strategies

As mentioned in Section 3.4.4, VICON can make predictions with varying timestep strides. Given I_0 initial frames $\{\mathbf{u}_i\}_{i=1}^{I_0}$, we can form in-context example pairs $\{\langle \mathbf{u}_i, \mathbf{u}_{i+s} \rangle\}_{i=1}^{I_0-s}$. For $j \geq 1$, we set the question condition ($\boldsymbol{c}_k$ in Equation (3.3)) as $\boldsymbol{u}_j$ to predict $\boldsymbol{u}_{j+s}$, enabling s-step prediction.

For long-term rollout, we employ VICON in an autoregressive fashion. The ability to predict with different timestep strides enables various rollout strategies. We explore two natural approaches: single-step rollout and flexible-step rollout, where the latter advances in larger strides to reduce the number of rollout steps.

For single-step rollout, we simply follow the autoregressive procedure with $s = 1$. Flexible-step rollout involves a more sophisticated approach: given a maximum prediction stride $s_{\max}$, we first make sequential predictions with a gradually growing $s = 1$ to $s_{\max}$ using $\mathbf{u}_{I_0}$ as the question condition, obtaining $\{\mathbf{u}_{I_0+s}\}_{s=1}^{s_{\max}}$. Using each frame in this sequence as a question condition, we then make consistent $s_{\max}$-step predictions while preserving all intermediate frames. This flexible-step strategy follows the approach in [YO24].

This strategy is also applicable to imperfect measurement cases where partial input frames are missing, e.g., due to sensor device error. In this case, we can still form pairs from the remaining frames, with minor adjustments to the strategy generation algorithms, as shown in Figure 3.1(c).

More details (including algorithms and examples) on rollout strategies are provided in Section 3.C.1 (for perfect measurements) and in Section 3.C.2 (for imperfect measurements).

3.4.6 Evaluation Metric

We evaluate the rollout accuracy of VICON using two different strategies described in Section 3.4.5. The evaluation uses relative and absolute L^2 errors between the predicted and ground truth frames, starting from the frame $I_0 + 1$. For relative scaling coefficients, we use channel-wise scaling standard deviation σ of ground truth frames, which vary between different prompts.

3.5 Experimental Results

We benchmark VICON against two state-of-the-art sequence-to-sequence models:
DPOT [HSL$^+$24] (122M parameters) and MPP [MRSBP$^+$23] (AViT-B architecture, 116M parameters), as well as Specialist U-Net baselines [GB22] trained individually per dataset. Implementation details for all baselines are provided in Section 3.B. The vanilla ICON, which would require processing over 114K tokens per frame (128×128×7), exceeds our available GPU memory and is thus computationally infeasible for direct comparison on these datasets.

For evaluation, we use both absolute and relative L^2 RMSE (Section 3.4.6) on rollout predictions. Since VICON offers the flexibility of predictions with different timestep strides, we evaluate our single trained model using both single-step and flexible-step rollout strategies (Section 3.4.5) during inference.

For conciseness, we present summarized plots and tables in the main text, and defer complete results and visualizations to Section 3.D. Ablation studies examining key design choices of VICON—including patch resolution, positional encoding, and context length, as well as an alternative CNN-based architecture—are presented in Section 3.D.1. The checkerboard artifacts (e.g., in Figure 3.9) of the PDEBench-Comp-HighVis dataset are discussed in Section 3.D.2.

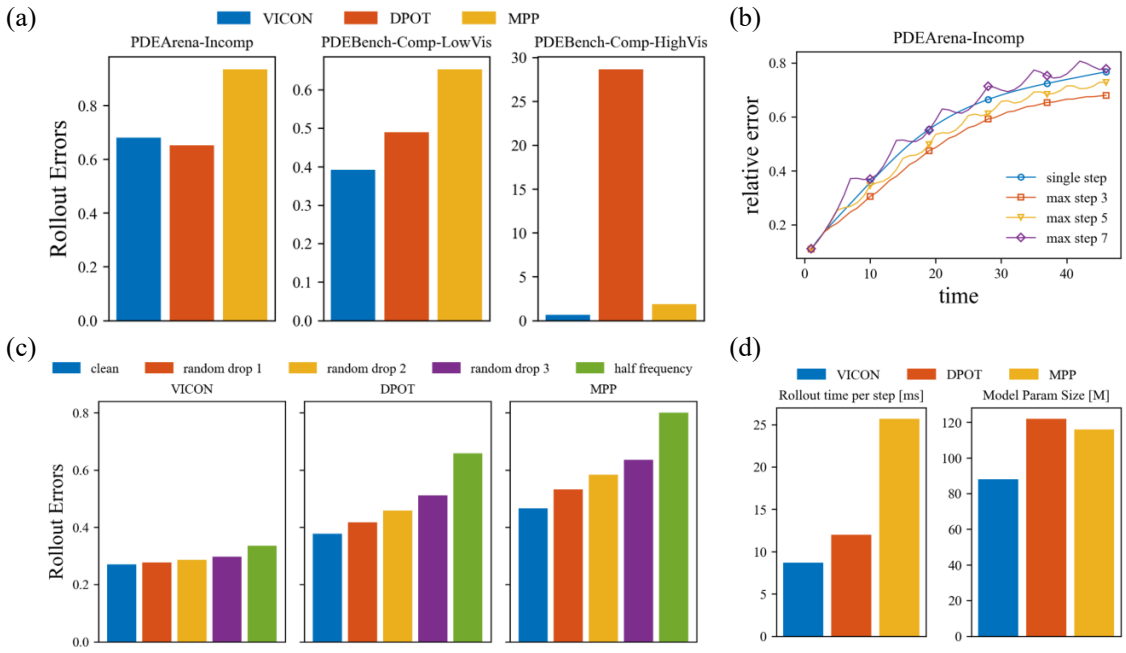


Figure 3.2: **Main experiment results.** (a) Last step rollout errors on 3 datasets. VICON outperforms MPP on all datasets and outperforms DPOT on 2 datasets. (b) VICON allows flexible rollout strategies to reduce error accumulation and demonstrates stride extrapolation. (c) VICON is robust to imperfect temporal measurements, while MPP and DPOT suffer from performance degradation. (d) VICON is smaller in size and has faster rollout time per step.

Table 3.1: **Summary of Rollout Relative L^2 Error (scaled by std)** across different methods and datasets. The best results are highlighted in bold. For flexible step rollout, step 3 works best for the PDEArena-Incomp dataset. Specialist U-Net is trained individually per dataset with instance normalization matching VICON's preprocessing.

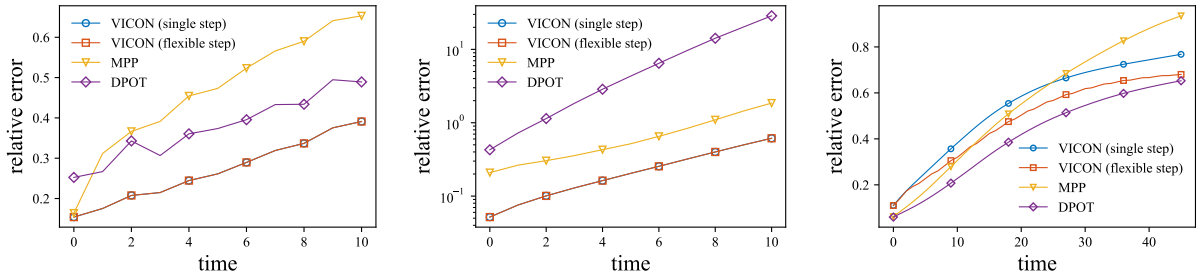
Rollout Relative L^2 Error	Case	Ours (single step)	Ours (flexible step)	DPOT	MPP	Specialist U-Net
Last step [1e-2]	PDEArena-Incomp	76.77	68.03	65.27	93.52	**48.8**
	PDEBench-Comp-LowVis	**39.11**	**39.11**	48.92	65.32	55.9
	PDEBench-Comp-HighVis	61.41	61.41	2866	185.3	**39.9**
All average [1e-2]	PDEArena-Incomp	56.27	48.50	41.20	55.95	**27.9**
	PDEBench-Comp-LowVis	**27.08**	**27.08**	37.72	46.68	35.7
	PDEBench-Comp-HighVis	30.06	30.06	821.9	72.37	**19.9**

3.5.1 Superior Performance on Long-term Rollout Predictions

As demonstrated in Figure 3.2(a), Table 3.1, and Figure 3.3, VICON consistently outperforms baseline methods on long-horizon predictions across all benchmarks—with the exception of DPOT on the PDEArena-Incomp dataset, where our performance is comparable. Overall, VICON achieves an average reduction in relative L^2 RMSE at the final timestep of 37.9% compared to DPOT and 44.7% compared to MPP.

Notably, DPOT exhibits exceptionally poor performance on PDEBench-Comp-HighVis, with an 821.9% error compared to our 30.06%. Our visualization of failure cases reveals that DPOT struggles with trajectories with small pressure values compared to the dataset average. This may stem from DPOT's lack of prompt normalization, as compressible flow's pressure channel exhibits large magnitude variations.

For the PDEArena-Incomp dataset, while VICON slightly underperforms DPOT and MPP initially, it quickly surpasses MPP and achieves comparable performance to DPOT for longer-step rollouts. This demonstrates VICON's robustness in long-term predictions. Despite DPOT's marginally better performance here, its poor performance on the PDEBench-Comp-HighVis dataset limits its applicability in multi-physics settings.



(a) PDEBench-Comp-LowVis (b) PDEBench-Comp-HighVis (c) PDEArena-Incomp

Figure 3.3: **Comparison of rollout errors (scaled by std) across different datasets and models.** We show errors for two VICON rollout strategies: single step rollout and flexible rollout strategy. For flexible step, step 3 works optimally for the PDEArena-Incomp dataset, while step 1 works best for PDEBench-Comp-LowVis and PDEBench-Comp-HighVis.

To further contextualize VICON's performance, we compare against Specialist U-Net baselines (U-Net-Mod64 from PDEArena [GB22]) trained individually per dataset. The only modification we made is adding instance normalization matching VICON's preprocessing, which we found essential for stabilizing training on PDEBench-Comp-LowVis and PDEBench-Comp-HighVis datasets. As shown in Table 3.1, while the specialist U-Net achieves strong performance on individual datasets it was optimized for, VICON remains competitive across all datasets and even outperforms the specialist on PDEBench-Comp-LowVis, demonstrating the advantage of our multi-physics approach in handling diverse physical systems without per-dataset architectural engineering or retraining.

More detailed results are provided in Table 3.13 and Table 3.14 in Section 3.D.

3.5.2 Flexible Rollout Strategies

As demonstrated in Figure 3.2(b) and Figure 3.5, VICON can select appropriate timestep strides based on each dataset's characteristics. While PDEBench-Comp-LowVis and

PDEBench-Comp-HighVis perform best with single-step rollout, PDEArena-Incomp achieves optimal results with a stride of 3 using our flexible-step rollout strategy.

This dataset-dependent performance reflects the characteristics of each dataset and the challenges of learning larger timestep strides. PDEBench-Comp-LowVis and PDEBench-Comp-HighVis record data with larger timestep sizes, making the learning of larger stride predictions more challenging due to increased dynamics complexity. Additionally, the fewer number of total rollout steps (even with stride=1) in these datasets cancels out the benefits of reducing rollouts when using multi-stride. Conversely, PDEArena-Incomp records data with smaller timesteps, making both single- and multi-stride prediction learning more feasible. Under this condition, multi-stride reduces the total rollout steps and error accumulation, achieving an 11.4% error reduction with a balanced stride ($s_{\max} = 3$).

When evaluated with unseen strides ($s_{\max} = 7$), VICON maintains comparable performance to single-step strategies for PDEArena-Incomp, indicating that it extracts underlying operators through context pairs rather than memorizing dynamics. This generalization capability provides tolerance when deployed to real-world settings where device sampling rates often differ from a fixed training set—an advantage we further demonstrate in the next section.

3.5.3 Importance of In-Context Learning

To validate the importance of in-context learning, we compare VICON against a non-in-context ViT baseline (with similar architecture: 162M parameters, 12 layers, hidden dimension 1024 for attention, and 4096 for FFN, time stride $\Delta t = 1$ for training) on the PDEArena-Incomp dataset. The results in Table 3.2 demonstrate that vanilla ViT cannot generalize to different timestep strides without in-context examples:

The significant performance degradation (95.03 vs 73.94 for last step error) demonstrates that vanilla ViT cannot generalize to different Δt without in-context examples, validating

Table 3.2: Non-in-context ViT baseline comparison on PDEArena-Incomp dataset.

Model	**Param Size**	**Stride 1 Error (1e-2)**	**Stride 2 Error (1e-2)**
VICON (Last step)	88 M	76.77	68.03
ViT (Last step)	162 M	73.94	95.03
VICON (All average)	88 M	56.27	48.50
ViT (All average)	162 M	50.17	63.15

our approach's effectiveness.

Additionally, we evaluated the impact of incorrect context pairs on VICON's performance using the PDEArena-Incomp dataset (Table 3.3).

Table 3.3: Impact of incorrect context pairs on VICON performance.

Experiment Setting	**Average Rollout Error**
Correct (stride 1)	0.55
Mix context pairs with stride 1 and 2	0.88
Context pairs with stride 2	0.92
Random noise context pairs	1.57

The clear performance degradation with incorrect context pairs confirms that VICON effectively utilizes in-context information rather than simply memorizing patterns.

3.5.4 Robustness to Imperfect Temporal Measurements

Real-world experimental measurements frequently suffer from imperfections—sampling rates may differ from the training set, and frames can be missing due to device errors. Sequence-

to-sequence models like MPP and DPOT require either retraining with new data or interpolation that introduces noise. In contrast, VICON elegantly handles such imperfections by forming context pairs from available frames only (Figure 3.1(c)). The algorithm for generating pairs with imperfect measurements appears in Section 3.C.2.

We evaluate VICON against baselines on all benchmarks under two scenarios: (1) half sampling rate (dropping every other frame) and (2) random frame dropping (removing 1-3 frames per sequence). Results for the PDEBench-Comp-LowVis dataset in Figure 3.2(c) (with complete results in Figure 3.8) show that VICON experiences only 24.41% relative performance degradation compared to 71.37%-74.49% in baselines—demonstrating remarkable robustness to measurement imperfections.

3.5.5 Turbulence Kinetic Energy Analysis

Turbulence kinetic energy (TKE), defined as $\frac{1}{2}(\overline{\tilde{u}_x^2} + \overline{\tilde{u}_y^2})$, is a critical metric for quantifying a model's ability to capture turbulent flow characteristics. Here, $\tilde{\boldsymbol{u}} = \boldsymbol{u} - \overline{\boldsymbol{u}}$ denotes the fluctuation of velocity from its statistical equilibrium state $\overline{\boldsymbol{u}}$.

Within our datasets, only a subset of PDEBench-Comp-LowVis, specifically those initialized with fully developed turbulent fields (see Appendix D.5 of [TPL+22]), is suitable for TKE analysis. After filtering these entries, we compare the mean absolute error (MAE) of the TKE between VICON, DPOT, and MPP. Our approach achieves a TKE error of 0.016, significantly outperforming MPP's error of 0.049, while achieving performance comparable to DPOT's error of 0.012. Visualizations of TKE errors are presented in Figure 3.6 in Section 3.D.

3.5.6 Benefits of Multi-Physics Joint Training

We empirically examine whether VICON benefits from multi-physics training by comparing two strategies: (1) training a single model *jointly* on all three datasets versus (2) training

separate models specialized for individual datasets. For fair comparison, we maintained identical batch sizes across all trainings while adjusting the steps for separate training to be slightly more than one-third of the joint training duration, ensuring **comparable total computational costs** between approaches.

As shown in Figure 3.7, joint training significantly outperforms separate training across all three datasets. We attribute this performance gain to the underlying physical similarities across these fluid dynamics problems, where exposure to diverse yet related flow patterns enhances the model's generalization ability.

3.5.7 Context Sampling Strategy

By default, VICON forms context pairs from initial frames within a single trajectory, which we term *intra-trajectory* context. This design choice is motivated by practical online deployment scenarios where only a few initial measurements are available, and system parameters (e.g., viscosity, forcing terms) may be unknown—making retrieval of historical trajectories with the same underlying operator infeasible.

As a theoretical question, we additionally investigate whether VICON can leverage *cross-trajectory* context, i.e., context pairs from different trajectories governed by the same operator. As detailed in Section 3.D.3, VICON trained with cross-trajectory context achieves similar performance with mixed contexts during rollout (only 0.012 increase in average error). This demonstrates the model's capability to leverage cross-trajectory information when available. Nevertheless, we argue that intra-trajectory context remains more practical as it avoids the need for maintaining trajectory databases and online operator identification.

3.5.8 Computational Efficiency

In Figure 3.2(d) and Table 3.16, we compare the computational resources required by VICON and baselines. Our model demonstrates superior efficiency across multiple met-

rics. Compared to MPP, VICON requires approximately one-third of the inference time per frame while using 75% of the total parameters. VICON also outperforms DPOT, requiring 28% fewer parameters and 28% less inference time.

It is important to note that, in theory, VICON incurs approximately 4× computational cost compared to non-paired, sequence-to-sequence baselines when all other settings (e.g., patch size) are identical, as it processes both condition and QoI, doubling the total token length with quadratic attention complexity. However, this computational overhead is justified by the significant performance improvements and unique capabilities (handling imperfect measurements, flexible timestep strides). Additionally, this cost can be reduced to 2× (linear) at inference time using KV caching, where all context pairs except the final pair remain fixed across rollout steps, allowing their key-value multiplications to be cached.

It is important to note that while post-training (such as fine-tuning) on a sequence-to-sequence model may achieve the same accuracy on one physical condition and save this 2× cost overhead, the finetuned model loses the ability to predict accurately when physical parameters may change on-the-fly during deployment, while VICON provides superior flexibility.

3.6 Conclusion and Future Work

We present VICON, a vision-transformer-based in-context operator network that efficiently processes dense physical fields through patch-wise operations. VICON overcomes the computational burden of original in-context operator networks in higher spatial dimensions while preserving the flexibility to extract multi-physics dynamics from few-shot contexts. Our comprehensive experiments demonstrate that VICON achieves superior performance in long-term predictions, reducing the average last-step rollout error by 37.9% compared to DPOT and 44.7% compared to MPP, while requiring only 72.5% and 34.8% of their respective inference times. The model supports flexible rollout strategies with varying timestep strides, enabling

natural application to imperfect real-world measurements where sampling frequencies differ or frames are randomly dropped—scenarios where VICON experiences only 24.41% relative performance degradation compared to 71.37%-74.49% degradation in baseline models.

Despite these advances, several challenges remain for future investigation. We empirically showed in Section 3.5.6 the benefits of multi-physics training, yet scaling to larger and more diverse datasets presents practical challenges. Specifically, generating high-fidelity physics simulation data across multiple domains requires substantial computational resources and specialized domain expertise including mesh generation, physical modeling, and numerical solver integration.

Furthermore, the current approach does not yet extend to 3D applications, as token sequence length would grow cubically, exceeding our computational budget. Potential remedies for 3D scaling include: (1) using VAE-like techniques/transformers to compress and evolve the fields in the latent space, then applying attention mechanisms to decode at any location [WSS+25, LSBF23], and (2) employing crop-like methods to learn evolution of fixed-width windows similar to CNNs. These approaches could potentially be combined to address the cubic scaling challenge.

Since VICON is an autoregressive model, it still suffers from error accumulation of the final pair's condition frame during rollout. Future work could explore resolving error accumulation through techniques such as injecting Gaussian noise or using push-forward methods [BWW21].

The channel-union approach is also limited when incorporating domains beyond fluid dynamics with fundamentally different state variables. Recent advances in inter-channel attention mechanisms [HLKT25] offer promising directions for handling channel scaling issues when expanding to more diverse physics systems.

Finally, adapting VICON to handle irregular domains such as graphs or meshes would broaden its applications to areas like solid mechanics and molecular dynamics. Techniques

such as geometric deep learning extensions, masked patch modeling, or point-cloud modeling are potential future directions to handle irregular geometries. The original ICON framework has demonstrated success across various problem types including inverse problems and steady-state tasks [YLMO23], suggesting potential for extending VICON beyond forward temporal prediction. Addressing these challenges will advance the promising paradigm of in-context learning for a broader range of physical systems.

3.A Dataset Details

3.A.1 PDEArena-Incomp Dataset

The incompressible Navier-Stokes dataset comes from PDEArena [GB22]. The data are generated from the equation

$$\partial_t \boldsymbol{u} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} = -\nabla p + \mu \Delta \boldsymbol{u} + \boldsymbol{f}, \tag{3.9}$$

$$\nabla \cdot \boldsymbol{u} = 0. \tag{3.10}$$

The space-time domain is $[0, 32]^2 \times [18, 105]$ where $dt = 1.5$ and the space resolution is 128×128. A scalar particle field is being transported with the fluids. The velocity fields satisfy Dirichlet boundary conditions, and the scalar field satisfies Neumann boundary conditions. The forcing term $\boldsymbol{f}$ is randomly sampled. The quantities of interest are the velocities and the scalar particle field.

3.A.2 PDEBench-Comp-HighVis and PDEBench-Comp-LowVis Datasets

The PDEBench-Comp-HighVis and PDEBench-Comp-LowVis datasets come from PDEBench Compressible Navier-Stokes dataset [TPL+22]. The data are generated from the equation

$$\partial_t \rho + \nabla \cdot (\rho \boldsymbol{u}) = 0, \tag{3.11}$$

$$\rho(\partial_t \boldsymbol{u} + \boldsymbol{u} \cdot \nabla \boldsymbol{u}) = -\nabla p + \eta \Delta \boldsymbol{u} + (\zeta + \eta/3)\nabla(\nabla \cdot \boldsymbol{u}), \tag{3.12}$$

$$\partial_t \left(\varepsilon + \frac{\rho u^2}{2} \right) = -\nabla \cdot \left(\left(\varepsilon + p + \frac{\rho u^2}{2} \right) \boldsymbol{u} - \boldsymbol{u} \cdot \sigma' \right). \tag{3.13}$$

The space-time domain is $\mathbb{T}^2 \times [0, 1]$ where $dt = 0.05$. The datasets contain different combinations of shear and bulk viscosities. We group the ones with larger viscosities into the PDEBench-Comp-HighVis dataset, and the ones with extremely small (1e-8) viscosities into the PDEBench-Comp-LowVis dataset. The PDEBench-Comp-HighVis dataset has space resolution 128×128. The PDEBench-Comp-LowVis dataset has raw space resolution 512×512 and is downsampled to 128×128 through average pooling for consistency. The quantities of interest are velocities, pressure, and density.

3.A.3 QOI Union and Channel Mask

Since each dataset contains different sets of quantities of interest, we take their union to create a unified representation. The unified physical field has 7 channels in total, with the following ordering:

1. Density (ρ)
2. Velocity in the x direction (u_x)
3. Velocity in the y direction (u_y)
4. Pressure (P)
5. Vorticity (ω)

6. Passively transported scalar field (S)

7. Node type indicator (0: interior node, 1: boundary node)

For each dataset, we use a channel mask to indicate its valid fields and only calculate loss on these channels. The node type channel is universally excluded from loss calculations across all datasets.

3.B Experiment Details

3.B.1 VICON Model Details

Here we provide key architectural parameters of VICON model implementation in Table 3.4.

3.B.2 Training Details

We implement our method in PyTorch [PGM+19] and utilize data parallel training [LZV+20] across two NVIDIA RTX 4090 GPUs. We employ the AdamW optimizer with a cosine learning rate schedule that includes a linear warmup phase. We apply gradient clipping with a maximum norm of 1.0 to ensure training stability. All optimization parameters are detailed in Table 3.5.

3.B.3 MPP Details

For a fair comparison, we retrain MPP [MRSBP+23] using our training dataset with the same model configurations and optimizer hyperparameters (batch size is set to the maximum possible on our device). We evaluate MPP using the same testing setup and metric.

Table 3.4: Model Configuration Details.

Patch Configuration	
Input patch numbers	8×8
Output patch numbers	8×8
Patch resolution	16×16
Positional Encodings Shapes	
Patch positional encodings	$[64, 1024]$
Function positional encodings	$[20, 1024]$
Transformer Configuration	
Hidden dimension	1024
Number of attention heads	8
Feedforward dimension	2048
Number of layers	10
Dropout rate	0.0
Number of COND & QOI pairs	10
Number of QOI exempted from loss calculation	5

Table 3.5: Optimization Hyperparameters.

Parameter	Value
Learning Rate Schedule	
Scheduler	Cosine Annealing with Linear Warmup
Peak learning rate	1×10^{-4}
Final learning rate	1×10^{-7}
Warmup steps	20,000
Total steps	200,000
Optimization Settings	
Optimizer	AdamW
Weight decay	1×10^{-4}
Gradient norm clip	1.0

3.B.4 DPOT Details

For a fair comparison, we retrain DPOT [HSL+24] using our training dataset with the same model configurations and optimizer hyperparameters (batch size is set to the maximum possible on our device). We evaluate DPOT using the same testing setup and metric.

3.B.5 Specialist U-Net Details

We use U-Net-Mod64, the strongest baseline from the PDEArena benchmark [GB22], trained individually per dataset. This architecture employs a U-Net with modified residual blocks and 64 base channels. We keep the architecture consistent across all datasets, only adjusting the input/output channel dimensions to match each dataset's state variables. We believe this is fair because in realistic deployment, one cannot architecturally re-engineer a model for every new measurement stream encountered.

Additionally, we include instance normalization matching VICON's preprocessing (channel-wise normalization using statistics computed from initial frames), which we found essential for stabilizing training on the PDEBench datasets (PDEBench-Comp-LowVis and PDEBench-Comp-HighVis). Without this normalization, training on compressible flows diverges due to the large magnitude variations in the pressure channel. The batch size is set to the maximum possible on our device. Each specialist model is trained on its respective dataset only, in contrast to VICON's joint multi-physics training.

3.C Algorithms and Examples of Strategy Generation

3.C.1 Strategy with Full Temporal Sequence

Algorithm for single-step strategy. Algorithm 1 presents our approach for generating single-step rollout strategies when all temporal frames are available. This strategy maintains a fixed stride of 1 between consecutive frames, providing stable but potentially error-accumulating predictions.

Algorithm for flexible-step strategy. Algorithm 2 describes our approach for generating flexible-step strategies with variable strides up to a maximum value. This approach reduces the total number of rollout steps required, potentially mitigating error accumulation for long sequences.

Rollout Strategy Example. We demonstrate two rollout strategies in Tables 3.6 and 3.7, showing single-step and flexible-step strategies, respectively. In both cases, the initial frames span from time step 0 to 9, and we aim to predict the trajectory up to time step 20.

As shown in Table 3.7, the flexible-step strategy initially uses smaller strides to build sufficient examples, then employs maximum strides for later rollouts, as detailed in Section 3.4.5. We note that repeated in-context examples appear in Table 3.7, which is common when the maximum stride is large and the initial frames cannot form enough examples. While the

Algorithm 1: GenerateSingleStepStrategy

Input: D: Number of demonstrations to use

R: Number of ground truth reference steps

T: Total steps in sequence

Output: S: Strategy list of example pairs and question pairs

```
S ← [ ]
Ei ← range(0, D)                    /* example input indices: 0,1,...,D-1 */
Eo ← range(1, D+1)                  /* example output indices: 1,2,...,D */
for i from R to T-1 do
    Qi ← i - 1                      /* question input is previous frame */
    Qo ← i                          /* question output is current frame */
    S.append((Ei, Eo), Qi, Qo)
end
return S
```

number of in-context examples in VICON is flexible and our model can accommodate fewer examples than the designed length, our preliminary experiments indicate that the model performs better with more in-context examples, even when some examples are repeated.

3.C.2 Strategy with Imperfect Temporal Sequence

When facing imperfect temporal sampling where certain frames are missing, we can still form valid demonstration pairs from the available frames. Algorithm 3 presents this adaptive pair selection process, which serves as a fundamental building block for all strategy generation algorithms in scenarios with irregular temporal data.

Algorithm for single-step strategy with drops. Algorithm 4 extends the single-step strategy to handle scenarios where frames are missing from the input sequence, adaptively forming strategies from available frames while maintaining the fixed stride constraint.

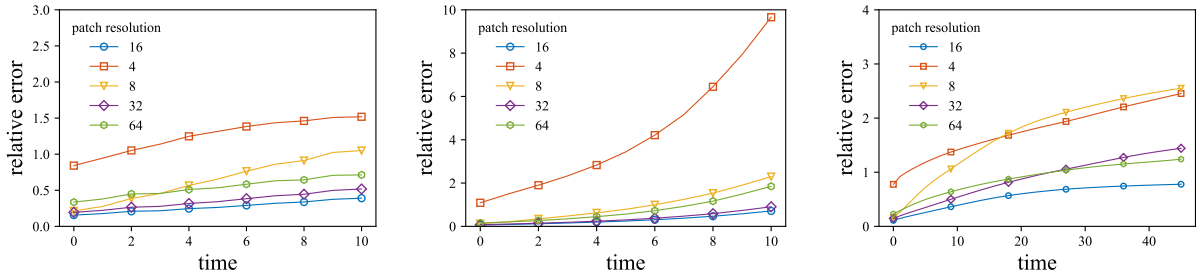
Algorithm for flexible-step strategy. Algorithm 5 presents our solution for generating flexible-step rollout strategies when frames are missing, dynamically selecting appropriate strides based on available frames and previous predictions.

Rollout Strategy Example with Missing Frames. Tables 3.8 and 3.9 illustrate our adaptive strategies when frames 2, 5, and 9 are missing from the initial sequence (with all other settings identical to those in Section 3.C.1). The tables demonstrate single-step and flexible-step (max stride: 3) approaches, respectively, highlighting how our algorithm dynamically selects appropriate example pairs to maintain prediction capability despite missing temporal data.

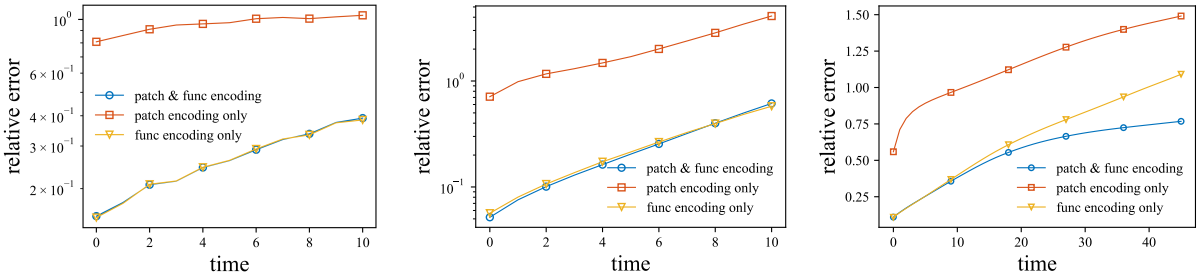
3.D Additional Experimental Results.

3.D.1 Ablation Studies

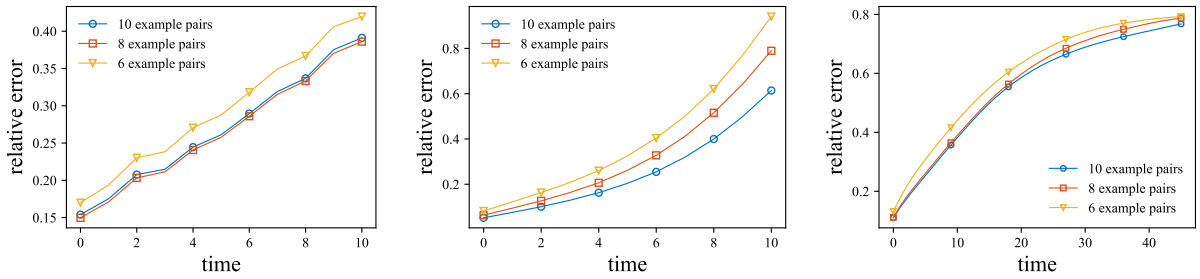
Impact of Patch Resolutions. We conducted ablation studies on patch resolution by varying patch sizes (4, 8, 16, 32, 64) to find a balance between spatial granularity and computational resource constraint. While smaller patches theoretically capture finer details, they generate longer token sequences, hitting memory caps due to transformer's quadratic complexity. For patch size 8, we reduced token dimensions (512→256) and feedforward dimensions (1024→512). For patch size 4, we applied further reductions (token dim 128, feedforward dim 256, layers 10→5) to fit within 80GB A100 memory—yet training still required 188 hours (vs 58 hours for ps=16) and inference slowed to 9.55 sec/step (vs 8.7 ms for ps=16). Patches below 4×4 remained infeasible even with these aggressive reductions. Figure 3.4(a-c) shows that patch size 16 achieves optimal performance across all datasets—balancing sequence length and representational capacity. Performance degradation with coarser patches (32, 64) aligns with expectations, while degradation with finer patches stems from the necessary reduction in hidden dimensions, highlighting a fundamental challenge in scaling to 3D applications.



(a) PDEBench-Comp-LowVis (b) PDEBench-Comp-HighVis (c) PDEArena-Incomp



(d) PDEBench-Comp-LowVis (e) PDEBench-Comp-HighVis (f) PDEArena-Incomp



(g) PDEBench-Comp-LowVis (h) PDEBench-Comp-HighVis (i) PDEArena-Incomp

Figure 3.4: Ablation studies across the three datasets. **Top row (a-c):** Impact of patch resolutions $(4, 8, 16, 32, 64)$ showing optimal performance at patch size 16. **Middle row (d-f):** Effect of different positional encoding combinations. **Bottom row (g-i):** Performance variation with different context lengths (6, 8, 10 pairs).

Positional Encodings. We evaluated different combinations of positional encodings as shown in Figure 3.4(d-f). Our architecture employs two types of encodings: patch position encodings (for spatial relationships between patches) and function encodings (differentiating input/output in different pairs) (Section 3.4.2).

The results demonstrate that including both encoding types consistently yields the best performance across all datasets. Notably, function encodings have a more significant impact than patch encodings, highlighting the importance of distinguishing between the condition and qoi beyond what causal masking ((3.6)) provides.

The impact of patch encodings varies across datasets: they show minimal effect on the PDEBench datasets (PDEBench-Comp-LowVis and PDEBench-Comp-HighVis) yet significantly improve performance on PDEArena-Incomp. We attribute this to the different stiffness of these systems. In compressible flows (PDEBench), spatial interactions naturally decay with distance, creating consistent and monotonic spatial correlations between patches. In contrast, Navier-Stokes equations exhibit infinite stiffness where correlations between any spatial locations are instantaneous and determined by the global velocity field, creating non-monotonic and case-dependent relationships. Explicitly encoding spatial positions therefore becomes particularly beneficial for PDEArena-Incomp, as it reduces the learning complexity for these non-local dependencies.

Varying Context Length. We investigated how the number of in-context examples affects model performance, as shown in Figure 3.4(g-i). Following insights from the original ICON work [YLMO23], we evaluated context lengths of 6, 8, and 10 pairs, balancing performance against computational efficiency. The results show that 10 in-context examples deliver significantly better results for PDEBench-Comp-LowVis compared to shorter contexts, while for the remaining datasets, 10 and 8 in-context examples perform similarly (both outperforming 6 examples).

Alternative Architecture: CNN-ICON. We explored an alternative architecture (CNN-ICON) that uses a CNN encoder to compress entire frames into single latent vectors, followed

by the ICON transformer structure. The encoder consists of a 4-level CNN with residual blocks ($7\times128\times128 \rightarrow$ 512-dim latent via global average pooling), paired with a symmetric CNN decoder. This results in 1 token per frame compared to VICON's 64 tokens (8×8 patches). We matched the model sizes: 86.5M parameters (vs VICON's 87.8M), identical transformer configuration (6 layers, 8 heads, dim=512), and same training protocol (200K steps).

As shown in Table 3.10, VICON outperforms CNN-ICON by 1.9–3.9$\times$ across all datasets. We attribute this performance gap to information loss when compressing entire frames into single vectors, which discards spatial structure crucial for accurate PDE prediction. This finding motivated our patch-based design, which preserves local spatial information while maintaining computational efficiency.

3.D.2 Analysis of Checkerboard Artifacts in PDEBench-Comp-HighVis

In our experiments, we observed block-like artifacts in the error/difference maps for the PDEBench-Comp-HighVis dataset that align with the model's 16×16 patch structure.
Through detailed analysis, we identified two contributing factors: 1) drastic field value variation across the trajectory time horizon, and 2) a training-inference normalization mismatch.

Drastic field value variation across trajectory time horizon. High-viscosity flows rapidly dissipate kinetic energy, causing velocity fields to become nearly stationary after approximately 10 frames. Our analysis of standard deviations across different time ranges reveals these differences:

Training-inference normalization mismatch. During training, context pairs are sampled uniformly across all frames. During inference, context pairs are sampled from initial frames only. As shown in Table 3.11, the standard deviations at these two different time ranges are vastly different, which results in normalization mismatch and exacerbates prediction errors in the already-challenging problem. Future work could explore using global

statistics from multiple trajectories instead of single-trajectory instance normalization during deployment. Other potential remedies include using overlapping patches or moving patch positions during rollout, though these represent engineering improvements orthogonal to our core contributions.

3.D.3 Cross-Trajectory Context Sampling

We investigate whether VICON can leverage context pairs from different trajectories governed by the same underlying operator, as suggested in the original ICON framework. Using the PDEArena-Incomp dataset (which provides 32 distinct trajectories per operator), we compare VICON trained with intra-trajectory context only versus a mixture of intra- and cross-trajectory context (50/50 split).

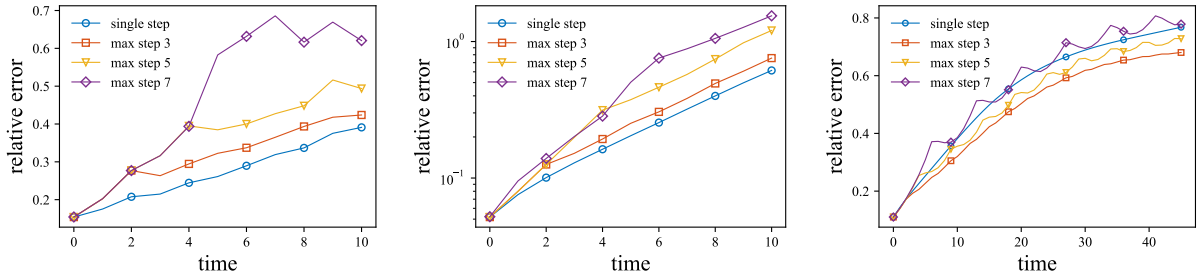
As shown in Table 3.12, VICON is capable of leveraging cross-trajectory information when trained appropriately. The model trained with mixed context achieves competitive performance in both settings, with only a marginal gap (0.556 vs 0.568) between intra- and cross-trajectory inference.

While these results demonstrate VICON's flexibility, we maintain that intra-trajectory context is more practical for deployment scenarios. Cross-trajectory sampling requires: (1) a pre-existing database of trajectories, and (2) a retrieval mechanism to ensure sampled trajectories share the same (often unknown) operator. In contrast, our intra-trajectory approach is self-contained and better suited for real-time forecasting where only the current trajectory is available.

3.D.4 More Results and Visualizations

Detailed Results. Tables 3.13 and 3.14 summarize the relative and absolute L^2 rollout errors across different timesteps for all evaluated models on the three datasets.

Generalization to different timestep strides. Figure 3.5 illustrates VICON's perfor-



(a) PDEBench-Comp-LowVis (b) PDEBench-Comp-HighVis (c) PDEArena-Incomp

Figure 3.5: Comparison of rollout errors across different datasets, using single-step and flexible-step strategies with varying maximum step sizes ($s_{\max} = 1, 3, 5, 7$).

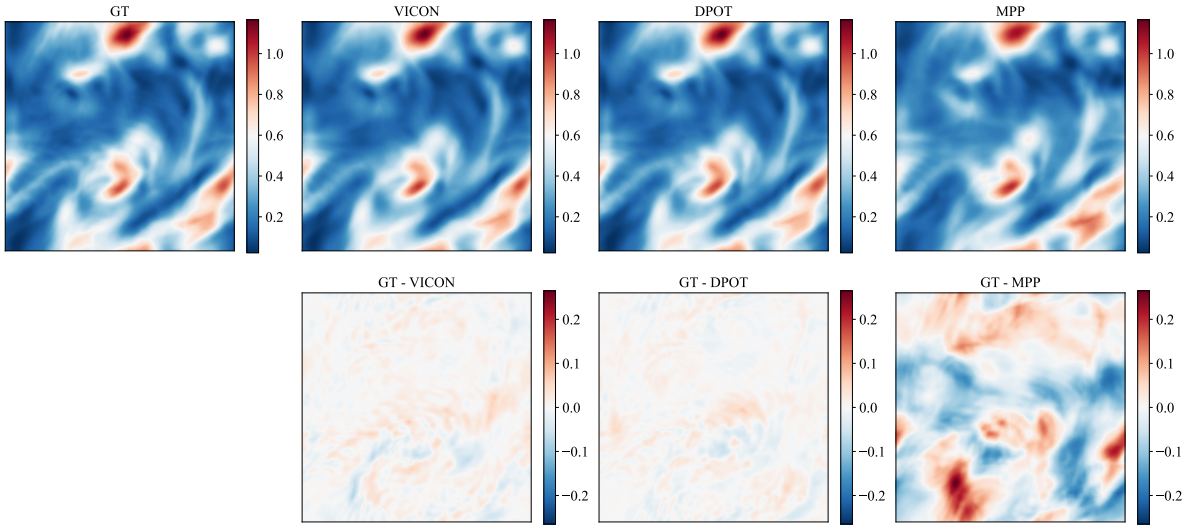
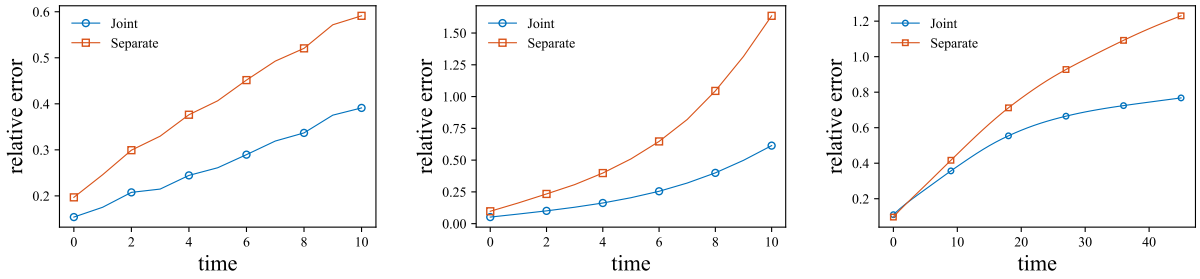


Figure 3.6: **Comparison of turbulence kinetic energy predictions.** (Top-left) Ground truth TKE field, (Top-right) model predictions, (Bottom) model error.

mance with varying timestep strides ($s_{\max} = 1, 3, 5, 7$) across all three datasets, demonstrating the model's ability to adapt to different temporal resolutions without retraining.

Turbulence Kinetic Energy (TKE) Predictions. Figure 3.6 displays the visualization of TKE fields for ground truth and model predictions, highlighting VICON's superior ability to preserve critical turbulent flow structures compared to baseline approaches.

Comparison between joint training vs separate training. Figure 3.7 and Table 3.15



(a) PDEBench-Comp-LowVis (b) PDEBench-Comp-HighVis (c) PDEArena-Incomp

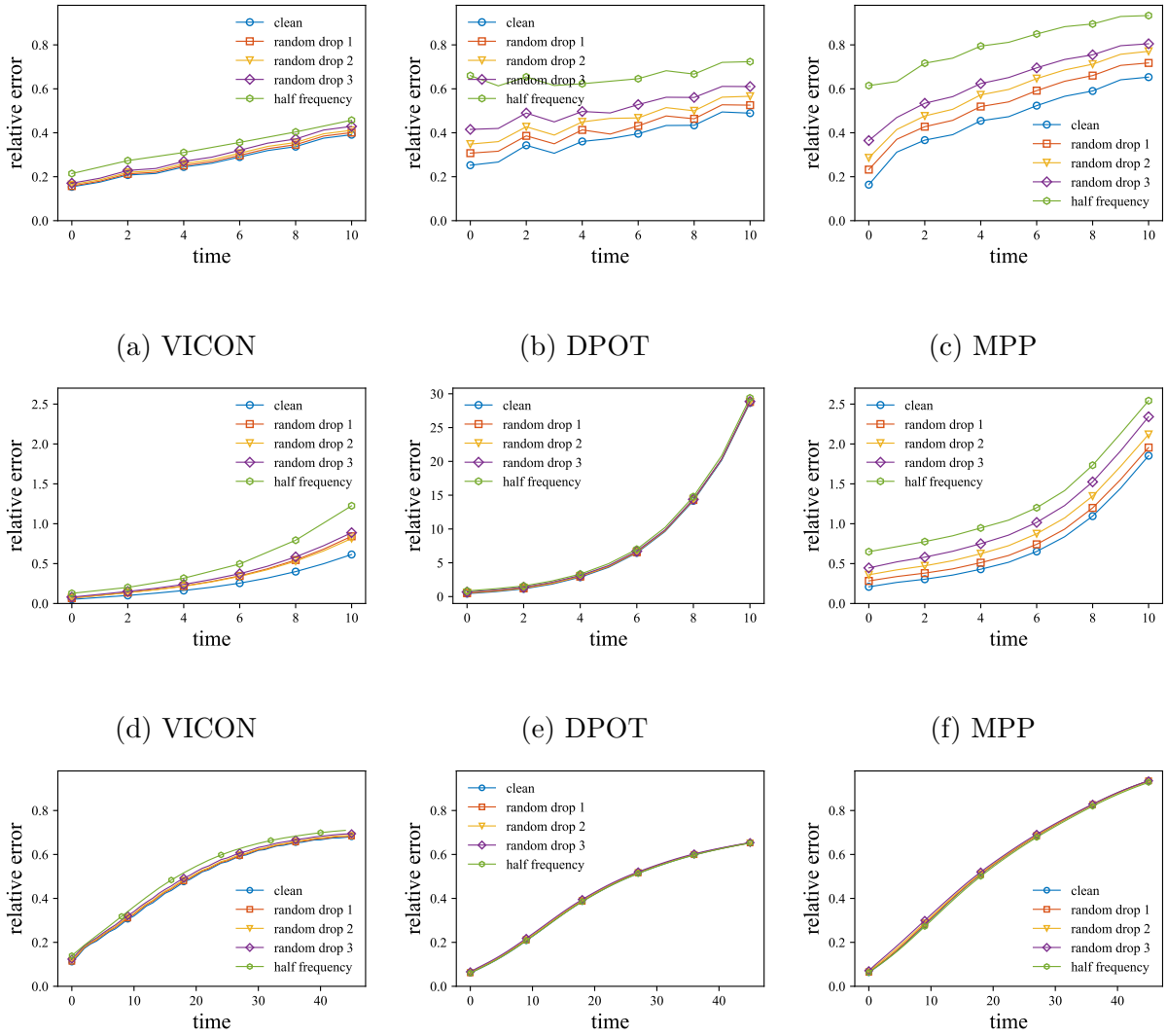
Figure 3.7: **Comparing rollout errors (single step, scale by std) for joint versus separate training strategies**. For separate training (individual models for each dataset), we maintain the same batch sizes as in joint training while adjusting training steps to be slightly more than one-third of the joint training duration, ensuring comparable total computational costs across both approaches.

quantify the performance differences between jointly trained and separately trained models, demonstrating the significant advantages of multi-physics training across all datasets.

Computational Efficiency. Table 3.16 summarizes the computational resources required by each method, showing VICON's advantages in terms of training cost, inference speed, and model parameter count compared to both DPOT and MPP baselines.

Rollout with imperfect measurements. Figure 3.8 show the rollout results with imperfect temporal measurements.

Visualizations. We compare the output of different models in Figure 3.9, Figure 3.10, and Figure 3.11. Figures 3.12 and 3.13 present additional visualizations of the VICON model outputs compared to ground truth and baseline predictions, highlighting the qualitative advantages of our approach.



(a) VICON (b) DPOT (c) MPP

(d) VICON (e) DPOT (f) MPP

(g) VICON (h) DPOT (i) MPP

Figure 3.8: **Comparison of rollout errors (scale by std) with different input data noise levels across all datasets.** Each column (left to right) shows VICON, DPOT, and MPP models, while each row (top to bottom) represents PDEBench-Comp-LowVis, PDEBench-Comp-HighVis, and PDEArena-Incomp datasets. For MPP and DPOT which require fixed *dt* and context window, interpolation is used to generate missing frames, while VICON can directly handle irregular temporal data without interpolation.

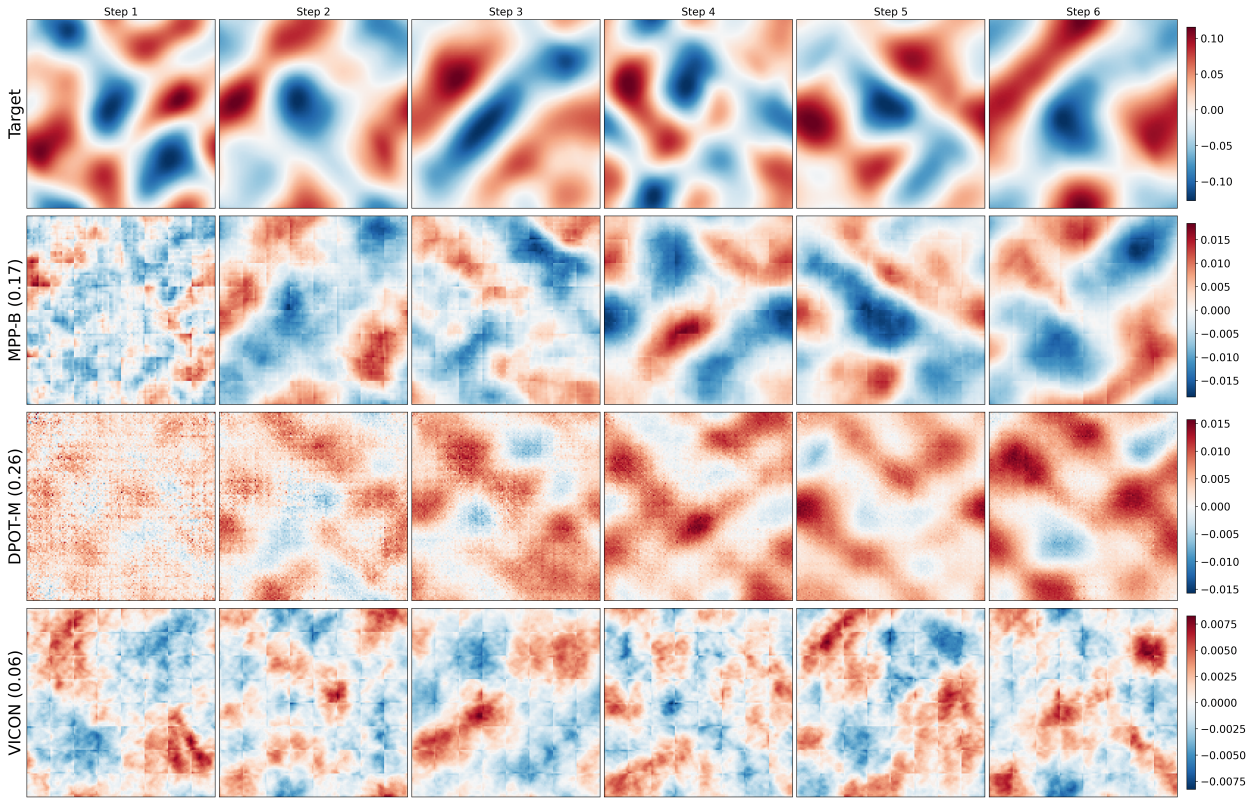


Figure 3.9: **Comparing outputs from different models.** The target is the first 6 output steps from PDEBench-Comp-HighVis dataset (x-velocity channel). For each model (each row), we display the difference between target and model output. Errors (rescaled by std) for the full trajectory are listed after the model names.

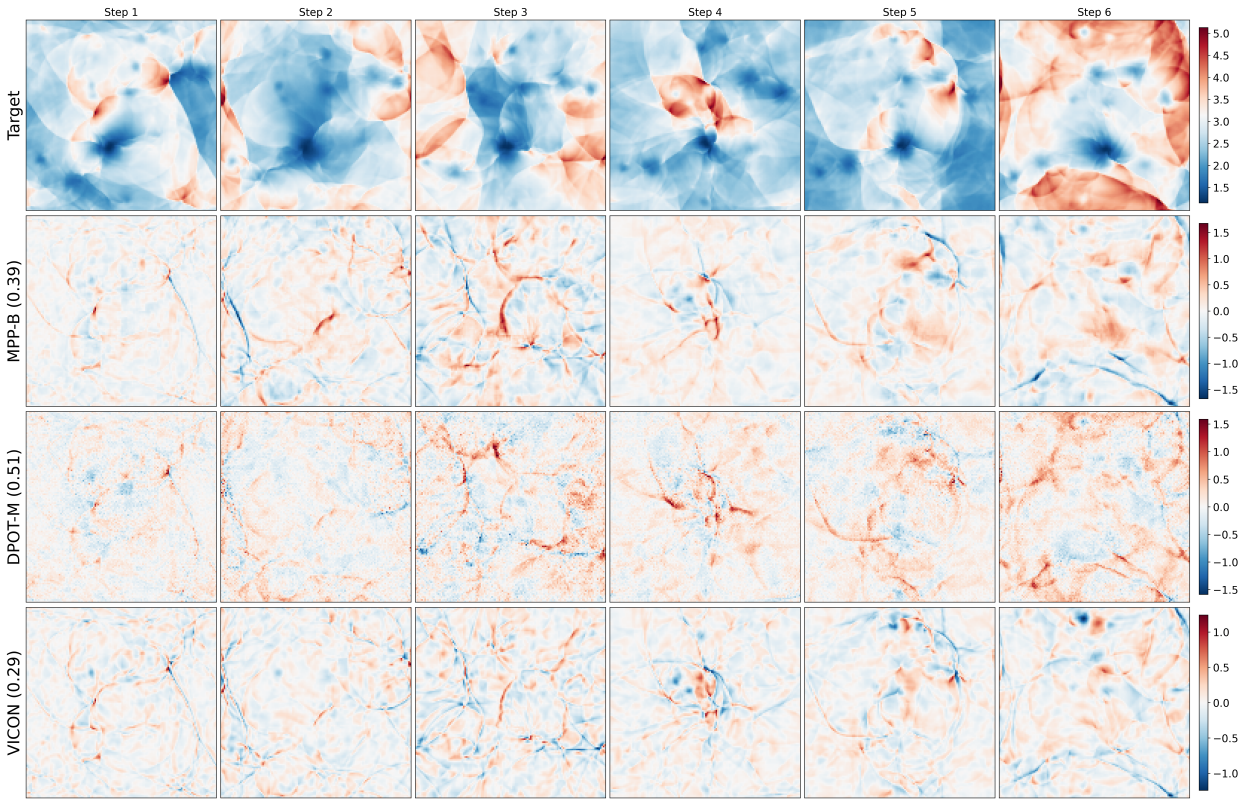


Figure 3.10: **Comparing outputs from different models.** The target is the first 6 output steps from PDEBench-Comp-LowVis dataset (pressure channel). For each model (each row), we display the difference between target and model output. Errors (rescaled by std) for the full trajectory are listed after the model names.

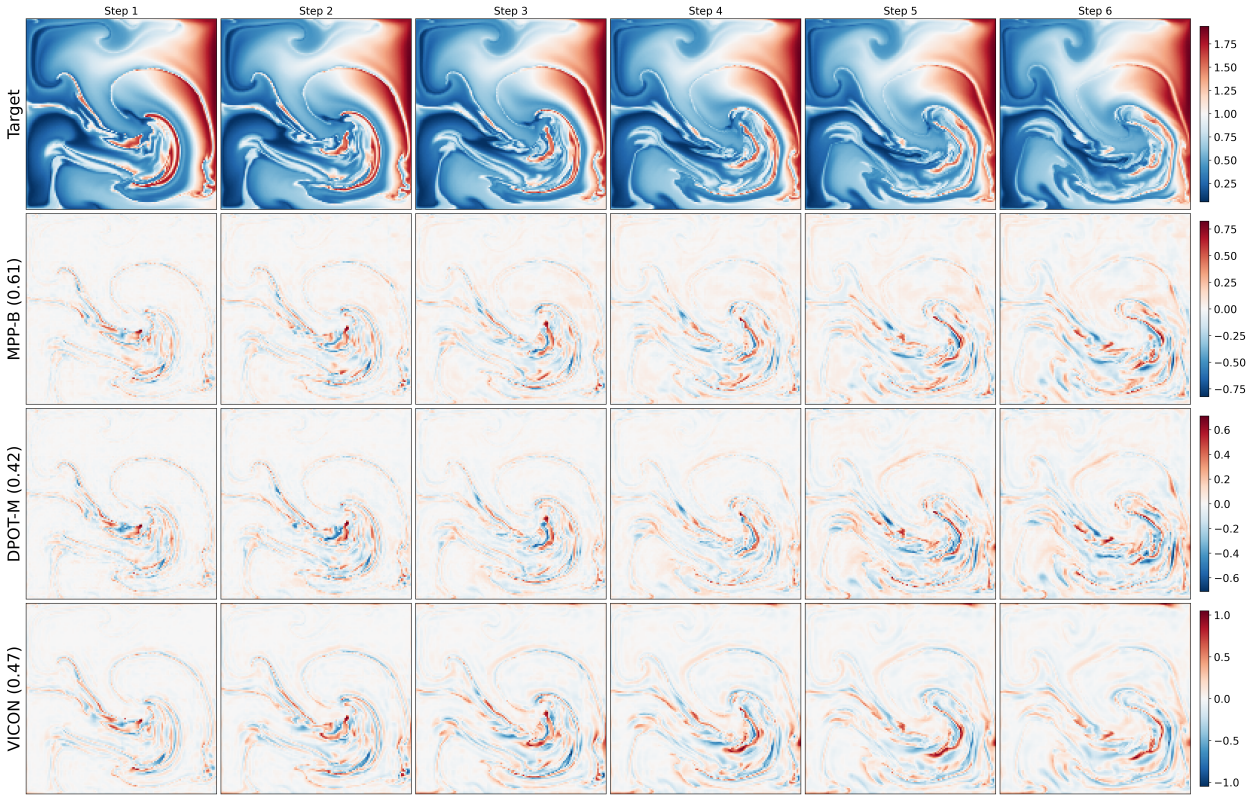
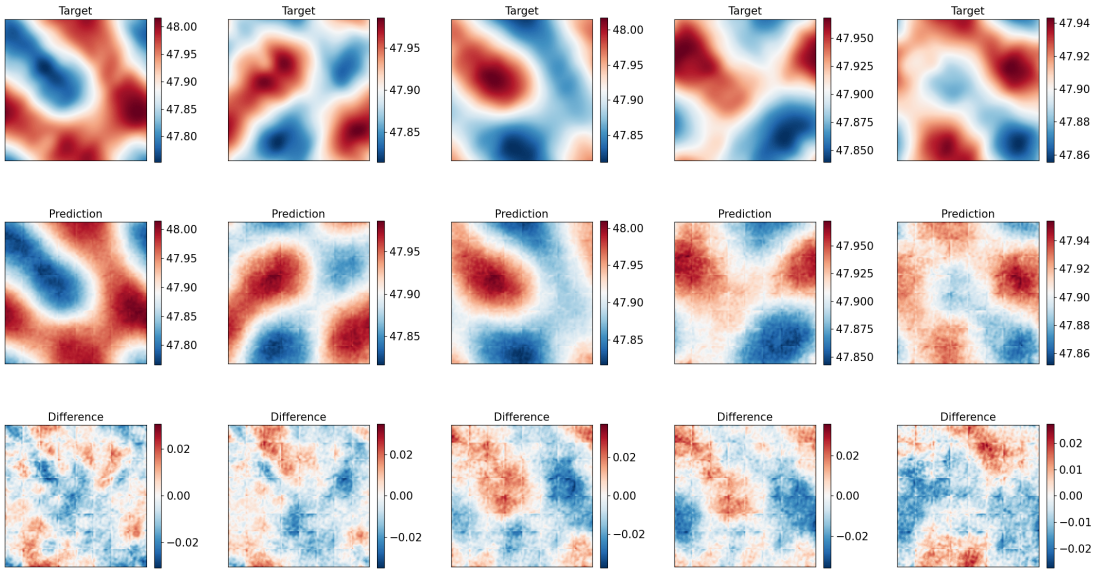
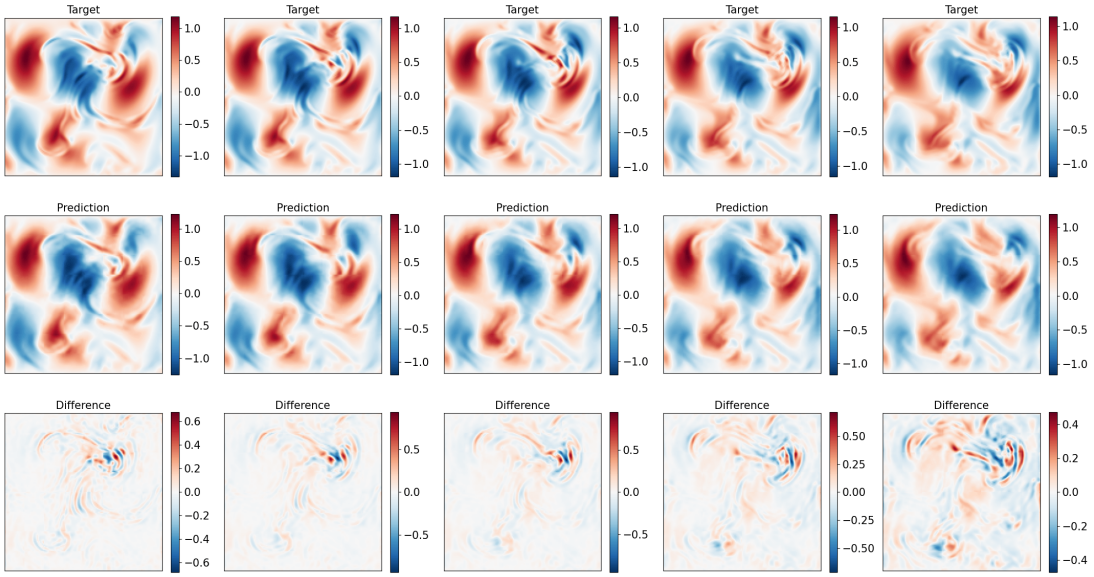


Figure 3.11: **Comparing outputs from different models.** The target is the first 6 output steps from PDEArena-Incomp dataset (particle density channel). For each model (each row), we display the difference between target and model output. Errors (rescaled by std) for the full trajectory are listed after the model names.

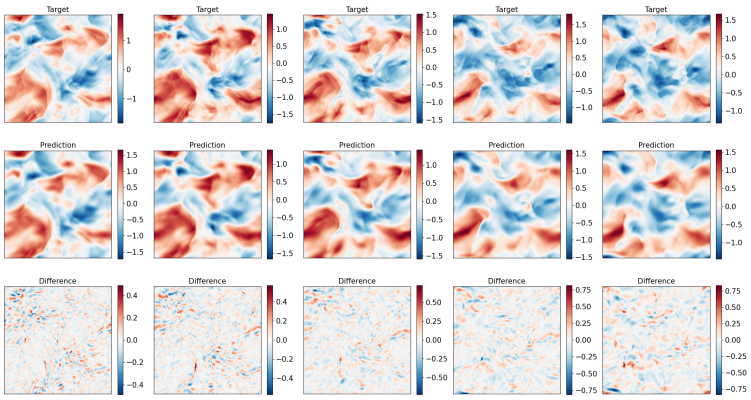


(a) A sequence of 5 output steps for PDEBench-Comp-HighVis dataset. The channel plotted is the pressure field in equation (3.12).

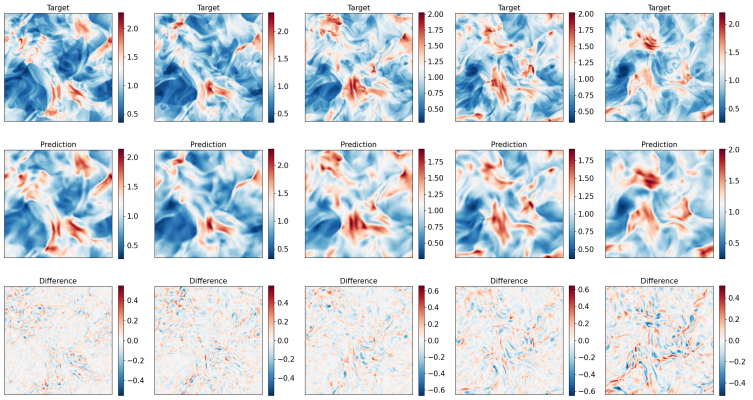


(b) A sequence of 5 output steps for PDEArena-Incomp dataset. The channel plotted is the x-velocity in equation (3.9).

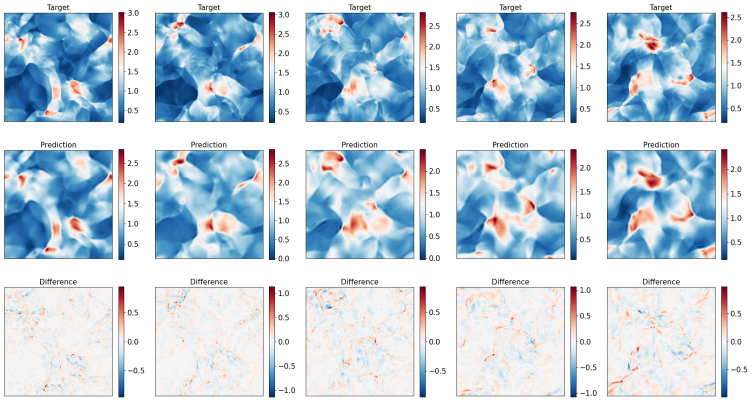
Figure 3.12: **Example outputs for the VICON model.** (a) The pressure field of PDEBench-Comp-HighVis dataset, and (b) the x-velocity field of PDEArena-Incomp dataset. Each column represents a different timestep.



(a) The channel plotted is the y-velocity field in equation (3.9).



(b) The channel plotted is the density field in equation (3.9).



(c) The channel plotted is the pressure field in equation (3.9).

Figure 3.13: **More example outputs for the VICON model.** Showing 5 output steps for the PDEBench-Comp-LowVis dataset as governed by equation (3.9): (a) y-velocity, (b) density, and (c) pressure fields. Each column represents a different timestep.

Algorithm 2: GenerateFlexibleStepStrategy

Input: D: Number of demonstrations to use

R: Number of ground truth reference steps

M: Maximum stride between pairs; T: Total steps in sequence

Output: S: Strategy list of example pairs and question pairs

```
/* Require R >= M + 1 to ensure examples exist for each stride      */
S ← [ ] Ec ← {} Eq ← {}        /* condition & query indices by stride */
/* Prepare example pairs for each stride                            */
for s from 1 to M do
    Nd ← min(D, R - s)           /* available examples for this stride */
    Eqo ← range(R - Nd, R)              /* output indices for examples */
    /* If we need more examples than available, repeat them         */
    reps ← ⌈D / Nd⌉                                 /* ceiling division */
    Eqo ← repeat(Eqo, reps)[0:D]             /* repeat and truncate to D */
    Eqo.sort()                                /* ensure increasing order */
    Ec[s] ← Eqo - s         /* input indices are s steps before outputs */
    Eq[s] ← Eqo                                  /* store output indices */
end
/* Generate rollout strategy                                        */
for i from R to T-1 do
    dist ← i - R + 1             /* distance from last reference frame */
    s ← min(dist, M)                        /* select appropriate stride */
    Qi ← i - s                                /* question input index */
    Qo ← i                                   /* question output index */
    S.append((Ec[s], Eq[s]), Qi, Qo)
end
return S
```

Table 3.6: Single-step Rollout Strategy Example

Rollout index	Examples (COND, QOI)	Question COND	Predict QOI
1	(0,1) (1,2) (2,3) (3,4) (4,5) (5,6) (6,7) (7,8) (8,9)	9	10
2	(0,1) (1,2) (2,3) (3,4) (4,5) (5,6) (6,7) (7,8) (8,9)	10	11
3	(0,1) (1,2) (2,3) (3,4) (4,5) (5,6) (6,7) (7,8) (8,9)	11	12
4	(0,1) (1,2) (2,3) (3,4) (4,5) (5,6) (6,7) (7,8) (8,9)	12	13
5	(0,1) (1,2) (2,3) (3,4) (4,5) (5,6) (6,7) (7,8) (8,9)	13	14
6	(0,1) (1,2) (2,3) (3,4) (4,5) (5,6) (6,7) (7,8) (8,9)	14	15
7	(0,1) (1,2) (2,3) (3,4) (4,5) (5,6) (6,7) (7,8) (8,9)	15	16
8	(0,1) (1,2) (2,3) (3,4) (4,5) (5,6) (6,7) (7,8) (8,9)	16	17
9	(0,1) (1,2) (2,3) (3,4) (4,5) (5,6) (6,7) (7,8) (8,9)	17	18
10	(0,1) (1,2) (2,3) (3,4) (4,5) (5,6) (6,7) (7,8) (8,9)	18	19
11	(0,1) (1,2) (2,3) (3,4) (4,5) (5,6) (6,7) (7,8) (8,9)	19	20

Table 3.7: Flexible-step Rollout Strategy Example ($s_{\max} = 5$)

Rollout index	Examples (COND, QOI)	Question COND	Predict QOI
1	(0,1) (1,2) (2,3) (3,4) (4,5) (5,6) (6,7) (7,8) (8,9)	9	10
2	(0,2) (0,2) (1,3) (2,4) (3,5) (4,6) (5,7) (6,8) (7,9)	9	11
3	(0,3) (0,3) (1,4) (1,4) (2,5) (3,6) (4,7) (5,8) (6,9)	9	12
4	(0,4) (0,4) (1,5) (1,5) (2,6) (2,6) (3,7) (4,8) (5,9)	9	13
5	(0,5) (0,5) (1,6) (1,6) (2,7) (2,7) (3,8) (3,8) (4,9)	9	14
6	(0,5) (0,5) (1,6) (1,6) (2,7) (2,7) (3,8) (3,8) (4,9)	10	15
7	(0,5) (0,5) (1,6) (1,6) (2,7) (2,7) (3,8) (3,8) (4,9)	11	16
8	(0,5) (0,5) (1,6) (1,6) (2,7) (2,7) (3,8) (3,8) (4,9)	12	17
9	(0,5) (0,5) (1,6) (1,6) (2,7) (2,7) (3,8) (3,8) (4,9)	13	18
10	(0,5) (0,5) (1,6) (1,6) (2,7) (2,7) (3,8) (3,8) (4,9)	14	19
11	(0,5) (0,5) (1,6) (1,6) (2,7) (2,7) (3,8) (3,8) (4,9)	15	20

Algorithm 3: GetAvailablePairs

Input: D: Number of demonstrations needed

dt: Desired time stride between pairs

Fa: List of indices of available frames

Output: P: List of input-output index pairs with the specified stride

```
Fa.sort()
P ← [ ]                                   /* Initialize empty pairs list */
/* Find all pairs with stride dt                                          */
for i from 0 to len(Fa) - 1 do
    for j from i + 1 to len(Fa) - 1 do
        if Fa[j] - Fa[i] == dt then
            P.append((Fa[i], Fa[j]))
        end
    end
end
if P is empty then
    return [ ]
end
/* If we don't have enough unique pairs, repeat them                      */
if len(P) < D then
    reps ← ⌈D / len(P)⌉                              /* ceiling division */
    RP ← repeat(P, reps)[0:D]          /* repeat pairs and truncate to D */
    P ← RP
end
return P
```

Algorithm 4: GenerateSingleStepStrategyWithDrops

Input: D: Number of demonstrations to use

S: Fixed stride between pairs

T: Total steps in sequence

Fa: List of indices of available frames

Output: St: Strategy list of (E, Qi, Qo) tuples

```
St ← [ ] Fa.sort()
Fs ← Fa[-1]                                  /* the starting frame for rollout */
P ← GetAvailablePairs(D, S, Fa)              /* pairs with fixed stride */
if P is empty then
    return [ ]          /* No available pairs with the specified stride */
end
Fc ← Fa.copy()            /* accumulated frames (current + predicted) */
for i from (Fs + 1) to (T - 1) do
    Ci ← i - S                               /* potential condition index */
    if Ci ∉ Fc then
        continue              /* Cannot predict frame i with stride S */
    end
    Pdt ← P[s]
                             /* The example pairs for predicting frame i */
    Qi ← i - s                               /* question input index */
    Qo ← i                                   /* question output index */
    S.append(Pdt, Qi, Qo)
    Fc.append(Qo)    /* The predicted frame can be used for future steps
     */
end
return St
```

Algorithm 5: GenerateFlexibleStepStrategyWithDrops

Input: D: Number of demonstrations to use
M: Maximum stride between pairs for rollout
T: Total steps in the sequence
Fa: List of indices of available frames
Output: S: Strategy list of (E, Qi, Qo) tuples, where E are the list of example pair indices (Ei, Eo) and Qi, Qo are the question and output indices

```
S ← [ ]; Fa.sort()
Fs ← Fa[-1]                                                    /* the starting frame for rollout */
P ← {}                                                         /* dictionary of available pairs */
for dt from 1 to M do
    Pdt ← GetAvailablePairs(D, dt, Fa)                         /* pairs with stride dt in Fa */
    if Pdt is not empty then
        P[dt] ← Pdt
    end
end
Ms ← max(P.keys())                                             /* maximum available stride */
Fc ← Fa.copy()                                                 /* accumulated frames (current + predicted) */
for i from (Fs + 1) to (T - 1) do
    dt ← i - Fs
    Mt ← min(dt, M, Ms)                                        /* maximum stride for current step */
    found ← False
    for s in P.keys().sorted(reverse=True) where s ≤ Mt do
        Ci ← i - s                                             /* potential condition index */
        if Ci ∈ Fc then
            /* Found a starting index to obtain frame i                                    */
            found ← True
            break
        end
    end
    if not found then
        /* We cannot predict frame i                                                      */
        continue
    end
    /* The example pairs for predicting frame i                                           */
    Pdt ← P[s]
    Qi ← i - s                                                 /* question input index */
    Qo ← i                                                     /* question output index */
    S.append(Pdt, Qi, Qo)
    Fc.append(Qo)
                                                               /* The Qo frame is obtained and can be used for future rollouts */
end
return S
```

Table 3.8: Single-step Rollout Strategy Example with Missing Frames

Rollout index	Examples (COND, QOI)	Question COND	Predict QOI
1	(0,1) (0,1) (0,1) (3,4) (3,4) (6,7) (6,7) (7,8) (7,8)	8	9
2	(0,1) (0,1) (0,1) (3,4) (3,4) (6,7) (6,7) (7,8) (7,8)	9	10
3	(0,1) (0,1) (0,1) (3,4) (3,4) (6,7) (6,7) (7,8) (7,8)	10	11
4	(0,1) (0,1) (0,1) (3,4) (3,4) (6,7) (6,7) (7,8) (7,8)	11	12
5	(0,1) (0,1) (0,1) (3,4) (3,4) (6,7) (6,7) (7,8) (7,8)	12	13
6	(0,1) (0,1) (0,1) (3,4) (3,4) (6,7) (6,7) (7,8) (7,8)	13	14
7	(0,1) (0,1) (0,1) (3,4) (3,4) (6,7) (6,7) (7,8) (7,8)	14	15
8	(0,1) (0,1) (0,1) (3,4) (3,4) (6,7) (6,7) (7,8) (7,8)	15	16
9	(0,1) (0,1) (0,1) (3,4) (3,4) (6,7) (6,7) (7,8) (7,8)	16	17
10	(0,1) (0,1) (0,1) (3,4) (3,4) (6,7) (6,7) (7,8) (7,8)	17	18
11	(0,1) (0,1) (0,1) (3,4) (3,4) (6,7) (6,7) (7,8) (7,8)	18	19

Table 3.9: Flexible-step Rollout Strategy Example with Missing Frames (max stride: 3)

Rollout index	Examples (COND, QOI)	Question COND	Predict QOI
1	(0,1) (0,1) (0,1) (3,4) (3,4) (6,7) (6,7) (7,8) (7,8)	8	9
2	(1,3) (1,3) (1,3) (4,6) (4,6) (4,6) (6,8) (6,8) (6,8)	8	10
3	(0,3) (0,3) (0,3) (1,4) (1,4) (3,6) (3,6) (4,7) (4,7)	8	11
4	(0,3) (0,3) (0,3) (1,4) (1,4) (3,6) (3,6) (4,7) (4,7)	9	12
5	(0,3) (0,3) (0,3) (1,4) (1,4) (3,6) (3,6) (4,7) (4,7)	10	13
6	(0,3) (0,3) (0,3) (1,4) (1,4) (3,6) (3,6) (4,7) (4,7)	11	14
7	(0,3) (0,3) (0,3) (1,4) (1,4) (3,6) (3,6) (4,7) (4,7)	12	15
8	(0,3) (0,3) (0,3) (1,4) (1,4) (3,6) (3,6) (4,7) (4,7)	13	16
9	(0,3) (0,3) (0,3) (1,4) (1,4) (3,6) (3,6) (4,7) (4,7)	14	17
10	(0,3) (0,3) (0,3) (1,4) (1,4) (3,6) (3,6) (4,7) (4,7)	15	18
11	(0,3) (0,3) (0,3) (1,4) (1,4) (3,6) (3,6) (4,7) (4,7)	16	19

Table 3.10: CNN-ICON vs VICON comparison (normalized RMSE, scaled by std)

Dataset	**Model**	**Step 1**	**Step 5**	**Step 10**	**Last**	**Avg**
PDEArena-Incomp	CNN-ICON	0.323	0.467	0.643	1.194	0.912
	VICON	0.110	0.206	0.305	0.680	0.485
PDEBench-Comp-LowVis	CNN-ICON	0.750	0.875	1.012	1.013	0.905
	VICON	0.154	0.245	0.375	0.391	0.271
PDEBench-Comp-HighVis	CNN-ICON	0.390	0.767	1.939	2.364	1.162
	VICON	0.052	0.163	0.498	0.614	0.301

Table 3.11: Standard deviation analysis across time phases for PDEBench datasets. The bolded velocity channels show PDEBench-Comp-HighVis has extreme variation between early and late phases, unlike PDEBench-Comp-LowVis with nearly consistent variation. This dynamic difference makes training a uniformed model particularly challenging.

Dataset	**Velocity u**			**Velocity v**			**Pressure**
	All	1-10	>10	All	1-10	>10	All
PDEBench-Comp-HighVis	0.359	0.511	**0.091**	0.351	0.506	**0.049**	14.254
PDEBench-Comp-LowVis	0.989	1.104	0.873	1.036	1.165	0.903	29.687

Table 3.12: Cross-trajectory context experiment on PDEArena-Incomp (normalized RMSE, scaled by std)

Training Setup	**Inference Setup**	**Avg Error**
VICON (trained with intra-traj only)	intra-traj context	0.563
	cross-traj context	0.609
VICON (trained with mixed context)	intra-traj context	0.556
	cross-traj context	0.568

Table 3.13: **(Comparison) Summary of Rollout Relative L^2 Error (scale by std)** for different methods across various cases. The best results are highlighted in bold.

Rollout Relative L^2 Error	**Case**	**Ours (single step)**	**Ours (flexible step)**	**DPOT**	**MPP**
Step 1 [1e-2]	PDEArena-Incomp	11.01	11.01	**5.97**	6.17
	PDEBench-Comp-LowVis	**15.40**	**15.40**	25.24	16.37
	PDEBench-Comp-HighVis	**5.18**	**5.18**	42.76	20.90
Step 5 [1e-2]	PDEArena-Incomp	22.68	20.62	**11.70**	14.81
	PDEBench-Comp-LowVis	**24.45**	**24.45**	36.05	45.45
	PDEBench-Comp-HighVis	**16.25**	**16.25**	286.3	42.93
Step 10 [1e-2]	PDEArena-Incomp	35.67	30.53	**20.74**	27.89
	PDEBench-Comp-LowVis	**37.54**	**37.54**	49.47	64.11
	PDEBench-Comp-HighVis	**49.83**	**49.83**	2016	144.46
Last step [1e-2]	PDEArena-Incomp	76.77	68.03	**65.27**	93.52
	PDEBench-Comp-LowVis	**39.11**	**39.11**	48.92	65.32
	PDEBench-Comp-HighVis	**61.41**	**61.41**	2866	185.3
All average [1e-2]	PDEArena-Incomp	56.27	48.50	**41.20**	55.95
	PDEBench-Comp-LowVis	**27.08**	**27.08**	37.72	46.68
	PDEBench-Comp-HighVis	**30.06**	**30.06**	821.9	72.37

Table 3.14: **(Comparison) Summary of Rollout Absolute L^2 Error** for different methods across various cases. The best results are highlighted in bold.

Rollout L^2 Error	**Case**	**Ours (single step)**	**Ours (flexible step)**	**DPOT**	**MPP**
Step 1 [1e-2]	PDEArena-Incomp	5.63	5.63	**3.02**	3.12
	PDEBench-Comp-LowVis	**21.74**	**21.74**	29.47	24.47
	PDEBench-Comp-HighVis	**1.43**	**1.43**	6.22	4.73
Step 5 [1e-2]	PDEArena-Incomp	10.38	9.46	**5.37**	6.79
	PDEBench-Comp-LowVis	**31.23**	**31.23**	43.57	57.50
	PDEBench-Comp-HighVis	**2.34**	**2.34**	14.48	6.52
Step 10 [1e-2]	PDEArena-Incomp	14.65	12.55	**8.55**	11.49
	PDEBench-Comp-LowVis	**45.39**	**45.39**	59.43	78.54
	PDEBench-Comp-HighVis	**3.21**	**3.21**	25.79	8.98
Last Step [1e-2]	PDEArena-Incomp	16.50	14.56	**14.03**	19.47
	PDEBench-Comp-LowVis	**48.69**	**48.69**	63.06	85.60
	PDEBench-Comp-HighVis	**3.39**	**3.39**	27.98	9.56
All average [1e-2]	PDEArena-Incomp	16.26	14.31	**11.86**	15.77
	PDEBench-Comp-LowVis	**34.44**	**34.44**	46.60	60.68
	PDEBench-Comp-HighVis	**2.48**	**2.48**	16.86	7.04

Table 3.15: **Summary of Rollout Relative L^2 Error Metrics (single step, scale by std) for 2 Training Strategies**. The best results are highlighted in bold. For separate training (a model on each dataset), each run's batch size is controlled to be the same as joint training. To ensure fair comparison, we maintain the same batch sizes as joint training for each separate training run while adjusting their training steps to be slightly more than one-third of the joint training duration, ensuring comparable total computational costs.

Rollout Relative L^2 Error [1e-2]	**Case**	**Joint**	**Separate**
Step 1	PDEArena-Incomp	11.10	**9.66**
	PDEBench-Comp-LowVis	**15.61**	19.68
	PDEBench-Comp-HighVis	**5.79**	9.74
Step 5	PDEArena-Incomp	**23.00**	24.09
	PDEBench-Comp-LowVis	**24.56**	37.63
	PDEBench-Comp-HighVis	**19.73**	39.81
Step 10	PDEArena-Incomp	**36.18**	41.66
	PDEBench-Comp-LowVis	**37.47**	57.17
	PDEBench-Comp-HighVis	**57.88**	131.6
Last Step	PDEArena-Incomp	**77.81**	123.0
	PDEBench-Comp-LowVis	**39.03**	59.11
	PDEBench-Comp-HighVis	**71.17**	163.5
All average	PDEArena-Incomp	**56.26**	76.35
	PDEBench-Comp-LowVis	**27.08**	40.75
	PDEBench-Comp-HighVis	**30.06**	65.19

Table 3.16: **(Comparison) Summary of Resource and Timing Metrics** for different methods.

Resource and Timing Metrics	**Ours**	**DPOT**	**MPP**
Training cost [GPU hrs]	58	70	64
Rollout time per step [ms]	8.7	12.0	25.7
Model Param Size	88M	122M	116M

CHAPTER 4

BCAT: A Block Causal Transformer for PDE Foundation Models for Fluid Dynamics

4.1 Introduction

Fluid mechanics is a fundamental area of study within physics and engineering, describing a wide range of phenomena through modeling of pressure, velocity, and viscosity. It is used in various applications, including climate forecasting, energy generation in wind and hydropower systems, aerodynamics and aircraft design, blood circulation analysis, and more. Due to the highly nonlinear interactions among multiple physical scales, predicting and simulating fluid behavior remains a challenging task. The computational problems become even more difficult in settings with limited observations or noisy measurements. Hence, accurately predicting fluid behavior in such regimes remains a challenge for scientific machine learning.

While foundation models like GPT [RNS+18, RWC+19, BMR+20], DALL-E [RPG+21, RDN+22], Stable Diffusion [RBL+22], LLAMA [TLI+23, TMS+23], Phi [AAA+24, AAB+24], Gemma [TRP+24], and Claude have demonstrated remarkable generalization capabilities in language and vision tasks [BHA+21], they have not accurately addressed scientific computing tasks such as simulating partial differential equations (PDEs). These tasks not only require precise predictions but also the ability to learn the underlying physical rules. PDE foundation models aim to address this challenge by using a single deep neural network (DNN) to simultaneously encode and approximate the behavior of a variety of physical

models. The objective is to develop a unified DNN capable of predicting multiple physical systems with the ability to simulate solutions in unseen parameter regimes, i.e. out-of-domain predictions. Recent progress in PDE foundation models includes the Predicting Operators and Symbolic Expressions (PROSE) [LZS24, SLZS25, SZS24, JSZS24, LSH+24], Multiple Physics Pretraining (MPP) [MRSBP+23], In-Context Operator Network (ICON) [YLMO23, YMLO23, YO24, CLY+26], DPOT [HSL+24], Fourier Forecasting Network (FourCastNet) [KSH+23], Poseidon [HRR+24], and Aurora [BBL+24].

Main Contributions: One of the key technical components of foundation models is the transformer architecture, known for its ability to efficiently model complex and long-range relationships in data through attention mechanisms. However, current transformer structures utilized in PDE foundation models are hindered by limited context and causality information, since many existing approaches learn to map a fixed window to the next step. In this work, we introduce the **BCAT** model, based on the **B**lock **CA**usal **T**ransformer architecture, adapting the attention mechanism for spatio-temporal dynamics modeling. The BCAT model performs next frame prediction instead of next token prediction, which enhances computational efficiency, allows for more token interactions to capture complex dependencies, and utilizes flexible history context windows. Our contributions are listed below.

- We introduce a new PDE foundation model based on a block causal transformer structure for auto-regressive prediction of fluid dynamics. BCAT utilizes next frame prediction to model spatiotemporal context with a 3.5x accuracy improvement over next token predictions. In our experiments, next frame prediction is 185x faster than next token prediction.

- BCAT's next frame prediction improves over next token prediction for efficient spatiotemporal modeling. Compared to fixed window update rules learned by previous SOTA methods, BCAT enjoys flexible context window, training parallelism through causality, and inference speed up via key-value cache.

- BCAT obtains state-of-the-art accuracy on widely used benchmark datasets: PDEBench, PDEArena, and CFDBench. Compared to larger SOTA models with 3.4 times the number of parameters as BCAT, BCAT reduces the error by 45%, demonstrating our model's parameter efficiency. Compared to similarly sized models, BCAT reduces the error by 66%, showing notable accuracy improvements over the current PDE foundation models. We demonstrate BCAT's transfer capability with limited training examples through fine-tuning.

4.2 Related Works

4.2.1 PDE Foundation Models

Foundation models are large-scale, pretrained models that act as flexible bases for a wide range of tasks, leveraging vast amounts of data and computational resources to enable adaptability and generalization across diverse applications [AAA+23, DJP+24, YYZ+24, BHA+21, BPH+24]. However, their applications to scientific computing are limited, where high precision is needed for numerical tasks. Recently, several approaches have been proposed to explore and develop foundation models for scientific computing. PROSE [LZS24, SLZS25, SZS24, LSH+24, JSZS24] takes multimodal input to construct solution operators, encoding PDEs as symbolic input to provide additional information and allows zero-shot generalization. ICON [YLMO23, YMLO23, YO24, CLY+26] uses an in-context learning approach, allowing the model to perform few-shot learning by observing input-output pairs. Some methods focus on the sequential modeling aspect, and various models have been designed to autoregressively predict future states. As additional information is not included, these methods are not able to zero-shot generalize to out-of-distribution data, but can leverage pretraining data to be efficiently finetuned [SHK+24]. Examples of such models include MPP [MRSBP+23], which uses axial attention to map a fixed context window to future states, and DPOT [HSL+24], which uses Fourier attention and stabilizes the autoregres-

sive rollout process through a denoising training process. Poseidon [HRR+24] is a swin-transformer-based model, and the model requires fine-tuning for downstream tasks.

4.2.2 Transformers for Time Series

Time series data takes a similar form as language modeling data, where numerical inputs replace discrete tokens. LLMs and other transformer-based models have been applied to various time series tasks, including forecasting [JWM+23], classification [LRM+21], and anomaly detection [ZY24]. One key challenge in adapting pretrained transformers for time series forecasting lies in the encoding of numerical data. In [GFQW24], scalar time series are encoded as strings, which can be directly processed by pretrained LLMs. In [CWPC24], a separate alignment stage is used to allow LLMs to process patched time series. However, as shown in [TMG+24], while attention layers can be useful for time series, pretrained LLMs may not outperform simple transformers trained from scratch. Another challenge lies in processing spatio-temporal dynamics, due to the complex interplay between spatial and temporal dependencies. Accurately capturing these interactions often requires models that can handle high-dimensional data, irregular sampling, and non-linear relationships, all while maintaining scalability and computational efficiency. Time-dependent PDEs, climate modeling, and videos can all be seen as examples of spatio-temporal data. The high-dimensional nature requires specific tokenization and processing techniques, such as graph [LYW+24] and Fourier mixer [KSH+23] for spatial dependencies.

4.3 Methods

4.3.1 Problem Setup

Consider parametric families of two-dimensional time-dependent nonlinear PDEs, whose state-variables of interest are represented by $\boldsymbol{u}(\boldsymbol{x}, t) \in \mathbb{R}^d$, where d can be up to 4 in our

tests, with $\boldsymbol{x} \in \Omega \subseteq \mathbb{R}^2$. Given data up to T_0 frames, i.e. the sequence:

$$\{\boldsymbol{u}(\cdot, t_i) \mid 0 \leq i < T_0\},$$

the goal is to predict the subsequent T frames:

$$\{\boldsymbol{u}(\cdot, t_i) \mid T_0 \leq i < T_0 + T\}.$$

In our experiments, we set $T_0 = T = 10$, that is, 10 frames are given as inputs and the model predicts 10 future frames.

4.3.2 Model Overview

The BCAT model is illustrated in Figure 4.1. The model comprises two main components: (1) image tokenization and de-tokenization through patch embedding, and (2) the standard decoder-only transformers similar to GPT-2 [RWC+19]. The input data, $\{\boldsymbol{u}(\cdot, t_i) \mid 0 \leq i < T_0\}$, are first partitioned into patches, which are subsequently flattened and projected into a sequence of visual tokens. The transformer backbone serves as a token mixer, processing these tokens to autoregressively predict future states $\{\boldsymbol{u}(\cdot, t_i) \mid T_0 \leq i < T_0+T\}$. We use RMS norm [ZS19] in place of the usual layer norms and find query-key normalization [HDPC20] helpful in stabilizing the training process.

Training is performed end-to-end by minimizing the mean squared error between the (normalized) predicted and ground truth sequences. Detailed descriptions of the two components are provided in the following sections.

4.3.3 Image Tokenization via Patch Embedding

Inspired by [DBK+21], we employ a patch embedding layer to convert the input data into a sequence of visual tokens. Each frame in the input data, $\boldsymbol{u}(\cdot, t_i)$, is first divided into non-overlapping patches, which are then flattened and projected into a sequence of visual tokens. For a maximum of $(T_0 + T)$ data frames, we simply concatenate the visual tokens from each

frame to form the input sequence to the transformer backbone.

Let R_x and R_y denote the space resolutions and c be the number of channels (physical fields), i.e., $\boldsymbol{u}(\cdot, t) \in \mathbb{R}^{R_x \times R_y \times c}$. In the experiments, we fix the spatial resolution to be $R_x = R_y = 128$, consistent with the majority of datasets considered. Data with different spatial resolutions are upsampled via interpolation or downsampled to match this resolution. To unify the number of channels (physical fields) across various PDEs, we zero-pad all input data to $c = 4$ to match the data size with the largest number of channels.

Choosing a patch size of P for each spatial dimension transforms each frame $\boldsymbol{u}(\cdot, t)$ into $(R_x/P) \times (R_y/P)$ patches, resulting in a sequence of up to $(T_0+T) \times (R_x/P) \times (R_y/P)$ visual tokens. We observed that the choice of patch size P significantly influences the performance of the model, which is quantified in Section 4.4.4. For our model, we select $P = 8$, the smallest size that maintains a reasonable sequence length. This choice results in a maximum sequence length of $(10 + 10) \times (128/8) \times (128/8) = 5120$.

After the transformer backbone, the sequence of tokens is projected back to physical space and then deflattened to reconstruct the predicted future states $\{\boldsymbol{u}(\cdot, t_i) \mid T_0 \leq i < T_0 + T\}$.

4.3.4 Preliminary: Next Token Prediction

In both language modeling and autoregressive image generation, next token prediction is the key learning task. For language models, the model predicts the probability distribution of the next token given the previous tokens, enabling the generation of coherent sentences by sampling tokens sequentially [VSP+17, RWC+19]. For autoregressive image generation, images are treated as sequences of pixels or patches, with each element predicted based on prior elements in the sequence [CRC+20]. Both applications rely on the autoregressive causal property, where future predictions depend only on prior context. More precisely, given a sequence of discrete tokens $X = (x_1, x_2, \ldots, x_S)$, the modeling assumption for most

language models is that:

$$p(x_1, \ldots, x_S) = \prod_{s=1}^{S} p(x_s | x_1, x_2, \ldots, x_{s-1}). \tag{4.1}$$

This is a reasonable assumption for sentence generation, as language typically follows a left-to-right sequential structure. However, for image generation, such dependencies are not as clear. Unlike text, images exhibit spatial invariance: the pixel/patch in the top-left corner does not inherently depend on the pixel/patch in the bottom-right corner.

4.3.5 Next Frame Prediction

While the causal property is less intuitive for spatial dependencies, it is a natural assumption for temporal dynamics, especially for Markovian sequences. Next frame prediction is the natural solution that addresses these concerns, where visual tokens within the same frame can attend to each other and all tokens for the same frame will be generated in parallel conditioned on previous frames.

Denote the input token sequence as $X = (x_1, \ldots, x_S)$ and the BCAT model as f_θ, where $S = (T + T_0) \times N$ is the total sequence length and $N = (R_x/P) \times (R_y/P)$ is the number of tokens per frame. Instead of learning to predict the immediate next token,

$$(x_1, x_2, \ldots, x_s) \xmapsto{f_\theta} (x_2, x_3, \ldots, x_{s+1}),$$

the BCAT model learns to predict the next frame given the previous frames:

$$(x_1, x_2, \ldots, x_s) \xmapsto{f_\theta} (x_{N+1}, x_{N+2}, \ldots, x_{s+N}).$$

To achieve this, similar to causal masks in transformers to mask out future tokens [VSP+17], we use a block lower triangular causal mask [CLY+26, TJY+24] to allow spa-

tial dependencies and temporal causality:

$$M = \begin{bmatrix} \mathbf{1}_{N\times N} & \mathbf{0}_{N\times N} & \mathbf{0}_{N\times N} & \dots & \mathbf{0}_{N\times N} \\ \mathbf{1}_{N\times N} & \mathbf{1}_{N\times N} & \mathbf{0}_{N\times N} & \dots & \mathbf{0}_{N\times N} \\ \mathbf{1}_{N\times N} & \mathbf{1}_{N\times N} & \mathbf{1}_{N\times N} & \dots & \mathbf{0}_{N\times N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{1}_{N\times N} & \mathbf{1}_{N\times N} & \mathbf{1}_{N\times N} & \dots & \mathbf{1}_{N\times N} \end{bmatrix}. \tag{4.2}$$

Here $\mathbf{1}_{N\times N}$ and $\mathbf{0}_{N\times N}$ represent constant 1/0 matrices of dimension $N \times N$.

We compare next frame prediction and next token prediction in Section 4.4.4, showing the accuracy gains from using next frame prediction in the prediction tasks. One additional benefit of next frame prediction over next token prediction is inference efficiency. For inference of autoregressive models, the model needs to be called repeatedly to generate the next token/frame sequentially. For next token prediction, to generate T future frames, the model needs to be called $T \times N$ times, while for next frame prediction, the model only needs to be called T times, providing a significant speedup (key-value caching can be used in both cases for further speedup). In our experiments, next frame prediction is 185x faster than next token prediction (see Appendix 4.4.5).

In [CLY+26, TJY+24], a similar block lower triangular mask (4.2) is used to perform "next function prediction" and "next scale prediction". However, the architecture, the tasks considered, and the learning objectives are different. VICON [CLY+26] is designed for contextual operator learning, while VAR [TJY+24] is designed for image generation. BCAT is optimized for spatiotemporal sequence modeling, focusing on different tasks. The only commonality is to allow bidirectional attention when causality does not hold, which naturally leads to the block causal mask.

Difference with fixed-window update rules. While methods such as MPP [MRSBP+23] and DPOT [HSL+24] also claim to do "next frame prediction," these methods in fact learn a fixed-windows update rule, which does not allow for flexible context windows. Additionally, without causality, these methods do not have training parallelism (one forward pass generates

only one output for loss computation) nor test time speed up (utilizing key-value cache is not possible). On the other hand, BCAT's next frame prediction improves over standard next token prediction for spatiotemporal modeling while still enjoying similar benefits: we can input the entire sequence for parallel training loss computation and use key-value cache to significantly speed up inference time.

4.3.6 Muon Optimizer

The Muon optimizer [JJB+24] is a variant of SGD with preconditioned momentum. Muon is designed to optimize 2D parameters of the hidden layers, and the preconditioning step orthogonalizes the momentum matrix using Newton-Schulz matrix iteration [Sch33]. When a network is trained with Muon optimizer, scalar/vector parameters of the network, as well as the input and output layers, should be optimized by a standard optimizer, such as AdamW [LH17].

Empirically, Muon demonstrated strong performance in training large language models [JJB+24], significantly outperforming AdamW in terms of convergence speed with $< 1\%$ computational overhead. More recently, [LSY+25] optimizes Muon's distributed implementation and studies the scaling law for training larger LLMs with Muon optimizer. Observing the success of Muon optimizer in training LLMs, we train BCAT using a combination of Muon optimizer (for transformer 2D parameters) and AdamW optimizer, which significantly outperforms BCAT trained only with AdamW optimizer. BCAT is the first model using Muon optimizer for scientific computing and prediction tasks, and we observe that Muon optimizer is able to better approximate high frequency or high contrast information. The comparison is shown in Section 4.4.4.

4.3.7 Implementation Details

The architecture utilizes a standard decoder-only transformer consisting of prenorm transformer blocks. We use RMS norm [ZS19] in place of layer norms and add query-key normalizations [HDPC20] after the attention projections, which we found to help stabilize the training process. In the feedforward network (FFN) of transformer layers, we use SwiGLU activation function [Sha20], and to keep a similar parameter count, the FFN hidden dimension is rescaled to be $\frac{8}{3}$x the feature dimension instead of the more common 4x. The patch embedding layer is efficiently implemented as a convolution layer with the same kernel size and stride (equivalent to MLP). More architecture and training details are included in Appendix 4.B.

4.4 Experiments

In this section, we first introduce the experiment setup and the baseline models we compare against. We present the main results comparing zero-shot performance in Section 4.4.2, and then study the transfer capability of PDE foundation models in Section 4.4.3. To demonstrate the efficiency of BCAT, we compare inference speed and GPU memory usage in Section 4.4.5. Various ablation studies for important architecture choices are included in Section 4.4.4. We include example visualizations of the BCAT model output and comparison of outputs from different models in Appendix 4.C.

4.4.1 Experiment Setup

Dataset. The dataset we use consists of 6 parametric families of PDEs that model fluid dynamics in different regimes (some families contain subfamilies using different ranges of physical parameters, so different data distributions exist even within a family). We collect the dataset from 3 heterogeneous sources: PDEBench [TPL$^+$22], PDEArena [GB22], and

CFDBench [LCZ23]. The dataset includes shallow water equations and the Navier-Stokes system with incompressible and compressible flow, regular and complex geometries, and different buoyancy settings. For datasets that do not provide a train/val/test splitting, we use the standard 80%/10%/10% splitting. The total amount of data samples is about 69K trajectory sequences (time homogeneity can be used to create more training input/output pairs), which is the current amount used in comparable models. More details are included in Appendix 4.A.

Evaluation Metric. We use the relative L^2 norm as the evaluation metric. More precisely, given the ground truth $\boldsymbol{u}$ and model's prediction $\tilde{\boldsymbol{u}}$, we compute the (time-averaged) relative L^2 error:

$$\frac{1}{T}\sum_{i=T_0}^{T_0+T-1}\frac{\|\boldsymbol{u}(\cdot,t_i)-\tilde{\boldsymbol{u}}(\cdot,t_i)\|_2}{\|\boldsymbol{u}(\cdot,t_i)\|_2+\varepsilon}, \tag{4.3}$$

where $T_0 = 10$ is the number of input steps, $T = 10$ is the number of output steps, and $\varepsilon = 10^{-7}$. Note that for the Navier-Stokes dataset from PDEArena, the temporal grid resolution is only 14, thus we set $T = 4$ for this dataset only. The average used in the last column of Table 4.1 is the average of the relative L^2 errors over the 6 families, i.e., the average over each row of the table.

Baselines and Comparisons. We evaluate our BCAT model against the following baseline methods. DeepONet [LJK19] and FNO [LKA+21] are two widely used single-operator learning techniques designed to efficiently approximate solution operators for PDEs. UNet [RFB15] is a classic convolutional neural network model for image processing, known for its symmetric hierarchical structure that enables it to capture both contextual information and fine details for pixel-wise predictions. ViT [DBK+21], a transformer-based image processing model, is notable for its ability to capture global dependencies in images and its scalability across various image and video processing tasks. Lastly, we test against three recent PDE foundation models, PROSE-FD [LSH+24], MPP [MRSBP+23], and DPOT [HSL+24]. PROSE-FD fuses data input and symbolic equation encoding for multimodal operator learning. Both MPP and DPOT autoregressively predict PDE solutions, with MPP based on

Axial-ViT and DPOT based on Adaptive Fourier Neural Operators.

Additional details about these baselines can be found in Appendix 4.B.3. We did not include Poseidon [HRR+24] as it does not generate meaningful results without fine-tuning, i.e., Poseidon cannot produce zero-shot predictions. Additionally, Poseidon cannot process history context, and thus is unable to handle partial observations of the state variables (e.g. the pressure and inhomogeneous forcing terms are not observed in PDEArena, but can be implicitly inferred from context). Some other transformer-based methods [LMF22, WLW+24] are not considered as their model sizes are much smaller, both MPP and DPOT have comparable or larger model sizes than BCAT. Note that DeepONet and FNO are designed for single tasks but can be extended to multi-task prediction using fine-tuning [SHK+24, ZML+24c]. Except Section 4.4.3 where we compare transfer capability, all other comparisons are performed in a zero-shot fashion, which is a standard metric also used by MPP and DPOT. As a reference, in Appendix 4.B.5, we train separate single-task models for each family, and the results show no significant difference.

4.4.2 Main Results

We present the main results in Table 4.1, where we report the relative L^2 error (%) for each family of equations and their average. We use the same zero-shot setting for all models: a single model is trained to predict all families of equations without any fine-tuning. The BCAT model achieves the best results in terms of average across all families, outperforming other models with various sizes, including MPP-L and DPOT-L which are $\sim$3x larger. For specific families, the BCAT model achieves the best results for five out of six families and the second-best results for the other family. For the one family which BCAT is the second-best model, it is outperformed by DPOT-L, which is 3.4x larger. In that case, most models have a relative L^2 error less than 1%. Notably, on the more complicated PDEArena NS-cond dataset where most baselines have higher errors, BCAT outperforms all other models with a nontrivial gap, demonstrating the effectiveness of our proposed model.

Table 4.1: **Main Results and Comparisons with Baselines**. The numbers reported are relative L^2 errors (%). The averages are taken with respect to the 6 distinct families listed in the columns of the table. We **bold** the best result in each column.

Model	Param	PDEBench			PDEArena		CFDBench	Average
		SWE	CNS*	INS	NS	NS-cond	-	
DeepONet	3.5M	3.55	7.41	64.61	35.33	51.85	12.50	29.21
FNO	0.6M	3.71	6.31	36.84	38.67	55.63	8.52	24.95
UNet	5.6M	0.33	3.19	3.43	12.56	16.82	0.76	6.18
MPP-B	116M	1.02	1.90	7.52	5.71	12.57	1.23	4.99
ViT	162M	0.25	1.49	2.82	7.05	12.41	0.55	4.10
MPP-L	407M	0.47	1.53	6.42	4.64	9.64	0.73	3.91
DPOT-M	122M	0.54	1.01	5.20	4.92	8.55	0.64	3.47
PROSE-FD	165M	0.28	1.41	2.75	5.27	9.61	0.61	3.32
DPOT-L	523M	0.15	0.89	4.08	2.21	5.29	**0.34**	2.16
BCAT	156M	**0.10**	**0.39**	**1.34**	**1.59**	**3.13**	0.52	**1.18**

Many datasets we considered have only 20 timestamps or fewer, which motivates our choice of 10 input steps and 10 output steps during evaluation. The more difficult PDEArena NS-cond dataset has a total of 56 timestamps with a relatively larger timestep of $dt = 1.5$. To study BCAT's long-time accumulation of error, we evaluate BCAT on all 56 timestamps for the PDEArena NS-cond dataset, and compare its performance with MPP-L and DPOT-L, the two largest models. To generate predictions beyond the training time window, for all three models, we use generated outputs as new initial conditions (i.e., rollout). In Figure 4.2, we plot the relative L^2 error per step for the three models. We observe that the error growth plateaus towards the end, likely due to the solutions nearing steady state. BCAT outperforms DPOT-L and MPP-L at all time steps.

4.4.3 Transfer Capability

We focus on and compare the zero-shot performance of the models considered, since zero-shot performance is an unbiased metric for the intrinsic capabilities of foundation models. For example, when GPT-2 [RWC+19] was introduced, only the zero-shot performance was studied. Additionally, it may be difficult to directly compare PDE foundation models finetuned with different algorithms, such as full/partial parameter finetune and LoRA [HSW+22]. Fine-tuning could obscure a model's weakness through overfitting on task-specific objectives.

However, for completeness, we include one experiment comparing the transfer capabilities of PDE foundation models. We use the same full parameter fine-tuning method for a fair comparison. The PDEBench Turbulence dataset is used, where the initial conditions are chosen to lead to turbulent flow. We finetune pretrained BCAT, DPOT-L, and MPP-L on the PDEBench Turbulence dataset for 500 gradient updates using a batch size of 64. Given the relatively small training dataset size of 1.6K, our training is equivalent to 20 epochs on the dataset, thus testing the low-data and low-compute regime which is typical with fine-tuning tasks. The results are shown in Table 4.2, where we observe that BCAT significantly outperforms larger models DPOT-L and MPP-L. As a reference, when we train BCAT model from scratch for the same 500 steps, the relative L^2 error is 29.95%, demonstrating how pretraining improves the result in this regime.

One potential mechanism that enables better transfer capabilities for BCAT compared to other approaches could be the next frame prediction. A fixed compute budget limits the number of gradient updates, which makes training parallelism essential for efficiency. As explained in Section 4.3.5, for fixed window update rules used by MPP and DPOT, each forward pass only generates 1 output step, resulting in 1 output for gradient update. BCAT's next frame prediction allows training parallelism through causality, and each forward pass generates multiple output steps for gradient update. This does not come with the additional cost of longer training time per step: total fine-tuning time for BCAT is around 10 minutes,

Table 4.2: **Comparing Transfer Capabilities on PDEBench Turbulence dataset.** Each model is finetuned for 500 gradient updates with a batch size of 64 (20 epochs).

Model	Param	Testing Relative L^2 Error (%)
MPP-L	407M	7.35
DPOT-L	523M	6.76
BCAT	156M	3.87

and total fine-tuning time for MPP-L and DPOT-L is more than 20 minutes (likely due to larger models).

4.4.4 Ablation Studies

In this section, we present the results of our ablation studies to validate some key optimizer and architecture choices.

Muon Optimizer vs. AdamW Optimizer We compare the BCAT model trained using a combination of Muon and AdamW optimizer (we refer to this as Muon for brevity) versus AdamW optimizer alone. When trained with Muon optimizer, the transformer layer matrix weights are optimized using Muon, and the other weights (scalar, vector, embedding, and initial/final projection layer weights) are optimized using AdamW, following [JJB+24]. We kept the same optimizer hyperparameters (originally tuned for AdamW), and only tuned the learning rate scheduler hyperparameters. We found that more decay steps help stabilize the training process for Muon.

The results are shown in Table 4.3, where Muon outperforms AdamW in all datasets, improving BCAT's performance in all cases. In terms of average errors, Muon reduces the error from 1.92% to 1.18%, at the cost of <1% computational overhead. We visualize and compare the output of BCAT trained using Muon and AdamW in Figure 4.3. We observe that, while the spatio-temporal error distributions are similar when trained with the two

Table 4.3: **Compare BCAT model trained with AdamW and Muon optimizer.** The numbers reported are relative L^2 errors (%). The averages are taken with respect to the 6 distinct families listed in the columns of the table.

Optimizer	Param	PDEBench			PDEArena		CFDBench	Average
		SWE	CNS*	INS	NS	NS-cond	-	
AdamW	156M	0.26	0.61	2.19	2.61	5.15	0.70	1.92
Muon	156M	0.10	0.39	1.34	1.59	3.13	0.52	1.18

optimizers (in high contrast and high frequency regions), the error magnitude is much lower when trained with Muon.

Next Frame Prediction vs. Next Token Prediction. We directly compare the performance of BCAT with next frame prediction and next token prediction on the PDEArena NS dataset. We use the same model and training setup for both models, except that the next-token-prediction variant is trained to predict the next token instead of the next frame (and the mask is causal instead of block causal). When trained on a mixture of all 6 families, the next-token-prediction variant fails to converge, so we only train it on the PDEArena NS dataset (the shorter dataset with 14 frames).

The results are shown in Table 4.4. We observe that the next-frame-prediction model achieves a 3.5x better relative L^2 error compared to the next-token-prediction variant, demonstrating the effectiveness of our proposed training objective. We also compute the error on the training dataset and the "next-token error." As the PDEArena NS dataset is a relatively small dataset, both models are overfitting to the training dataset to some degree; however, there is a performance difference between the two as shown by the error gap.

The "next-token error" is computed assuming all previous tokens are exactly known (i.e. no prior error). More precisely, during testing time, we input the full trajectory to the model with proper masks to prevent data leaking, similar to training time. Both models are only

Table 4.4: **Comparing next frame prediction and next token prediction.** The errors shown are relative L^2 errors (%) for PDEArena NS dataset. *The Next-Token error is computed assuming all previous tokens are exactly known (i.e., the full trajectory is inputed to the model, similar to training time). It is not a standard metric and is only used to provide a comparison.

Method	Param	Testing Error	Training Dataset Error	Next-Token Error*
Next token prediction	156M	5.49	1.62	1.01
Next frame prediction	156M	1.59	0.91	0.81

asked to predict the immediate next token/frame and thus no autoregressive rollout is needed. We are thus measuring the amount of error over one step, i.e., the "local error". In this case, we observe that the performance of both models are relatively similar. This shows that the two models have similar (local) approximation power, but the next-frame-prediction model is better conditioned for multi-step prediction. Specifically, the next-frame-prediction has a slower accumulation of error rate than the next-token-prediction approach. As discussed in Section 4.3.5, to autoregressively generate T future frames, the next-frame-prediction model only needs to be recursively called T times, while the next-token-prediction variant needs to be recursively called $T \times N$ times (where N is the number of tokens per frame). This may be the mechanism that leads to a significant reduction in error accumulation for the next-frame-prediction model.

Comparison with Spatio-Temporal Attention. The forward prediction task for 2D fluid dynamics is similar to a video generation task, where the state variable at each timestamp is akin to a frame in the video. In [BWT21], several different forms of self-attention for processing spatial-temporal dynamics are compared for video classification task. Our BCAT model flattens the patches from all frames into a single sequence, and can be seen as a variant of joint space-time self-attention discussed in [BWT21]. Similarly, MPP [MRSBP+23] is a variant of axial self-attention. Another approach that has not appeared in the literature as a

Table 4.5: **Comparing the performance of BCAT and Time-then-Space Model.** The errors shown are relative L^2 errors (%).

Model	Param	Average Testing Error
Time-then-Space	206M	1.19
BCAT	156M	1.18

backbone to PDE foundation models is the divided space-time self-attention, where temporal attention and spatial attention are separately applied one after the other. We implement this variant and refer to it as the Time-then-Space model. The temporal attention is a causal self-attention and the spatial attention is a full self-attention. All other configurations and hyperparameters are kept the same as the BCAT model. Due to more attention layers, the Time-then-Space model is larger than BCAT (206M vs 156M). The results are shown in Table 4.5, where the BCAT model slightly outperforms the spatio-temporal variant. Consider the difference in model sizes, this shows that the joint space-time self-attention is more effective for the forward prediction task.

Effect of Patch Size. In the image tokenization step, the patch size P to divide the image into non-overlapping patches is a key hyperparameter. Note that this is a tradeoff between model complexity and feature spatial resolution, as half the patch size makes the sequence length 4x longer, thus 16x the time complexity and 4x space complexity (assuming memory efficient attention), even though the model parameter size remains similar, since models with smaller patch size have fewer parameters in patch embedding due to smaller dimensions per patch. However, we observe that having a smaller patch size and hence more visual tokens significantly improves the performance for both BCAT and ViT. The results are shown in Table 4.6. For patch size 4 or below, we refer to [WYW+25] for a more comprehensive study on patch size scaling law (note that linear attention is used when the sequence length is beyond 4000).

Table 4.6: **Comparing the performance of BCAT and ViT with different patch size.** The errors shown are relative L^2 errors (%).

Model	Patch Size	Param	Average Testing Error
ViT	16	162M	4.10
ViT	8	155M	3.34
BCAT	32	191M	3.61
BCAT	16	163M	1.99
BCAT	8	156M	1.18

Additional Architecture Ablation Study. We validate some architecture details by comparing the average testing error. The results are shown in Table 4.7. We found query-key normalization [HDPC20] helpful in stabilizing the training process, especially towards the end of the training phase. Notably, when the Muon optimizer is used, the training loss diverges without query-key normalization, demonstrating the effectiveness of this technique. This also suggests that the Muon optimizer has different optimization dynamics for query and key projection matrices. For this ablation experiment only, we report the result using the AdamW optimizer. Under similar parameter counts, we found SwiGLU activation function [Sha20] better than GeLU. Note that the parameter count is not exactly the same since the MLP hidden dimension is rounded to the nearest multiple of 64. For the attention mask, to prevent data leaking in next frame prediction, any submask of the block causal mask (4.2) can work. We compare the BCAT model and a variant using a standard causal mask to perform next frame prediction. We observe that the block causal mask leads to better performance, demonstrating that having more token interactions within the same frame is beneficial for the forward prediction task.

Table 4.7: **Additional Ablation Studies.** Each row represents a different variant and we underline the difference with the BCAT model (last row). The errors shown are relative L^2 errors (%). *The training loss diverges when using Muon without QK-Norm, thus for this row only, we report the result when trained using AdamW.

Activation	QK-Norm	Mask	Param	Average Testing Error
SwiGLU	No	block causal	155.6M	2.04*
SwiGLU	Yes	causal	155.6M	1.97
GeLU	Yes	block causal	154.8M	1.34
SwiGLU	Yes	block causal	155.6M	1.18

4.4.5 Resource Usage Comparison

In Table 4.8, we compare the inference speed and GPU memory usage for all models. To obtain these results, on a single NVIDIA H100 GPU, we use a batch size of 1 and measure the time required to generate 10 output steps for the shallow water equation. To ensure stability, we average run time over 50 repeats after some warmup. While smaller models (FNO, DeepONet, and UNet) have a faster run time, BCAT outperforms all models with comparable or larger sizes except ViT, which shares the same transformer layer without autoregressive rollout.

4.4.6 Experimental Summary and Model Novelty

We summarize the experimental results and highlight the significance of the BCAT model.

Performance. BCAT significantly improves over previous SOTA in terms of both zero-shot prediction and transfer capability. Compared to 3.4 times larger model, BCAT reduces the error by 45%, and compared to similar-sized models, BCAT reduces the error by 66%.

Novel learning objective. BCAT's next frame prediction enhances scientific machine

Table 4.8: **Comparing the inference speed and GPU memory usage for all models.** We measure the resource usage for different models and variants to generate 10 output steps for a single trajectory.

Model	Param	Inference Speed (ms)	GPU Memory Usage (MB)
DeepONet	3.5M	2.7	190
FNO	0.6M	1.7	372
UNet	5.6M	1.3	135
ViT	155M	5.4	2362
MPP-B	116M	207.9	3087
MPP-L	407M	438.3	10869
DPOT-M	122M	95.7	1864
DPOT-L	523M	194.3	7985
Time-then-Space Variant	206M	175.5	3168
Next-token-prediction Variant	156M	13165	2402
BCAT	156M	70.9	2499

learning by integrating the strengths of classical next token prediction with spatio-temporal modeling. It improves upon fixed window update rules for training parallelism and testing speed up. Additionally, the causality better aligns with the Markovian property of the PDEs considered.

Architecture improvements. Architecture improvements in BCAT are helpful for the broader community. (1) In Section 4.4.4, we show that decreasing the patch size improves the performance of both BCAT and ViT. (2) We drop-in replaced PROSE-FD's optimizer by Muon, and the performance improved from 3.32% to 3.04%. (3) Compared to standard activation functions like ReLU and GeLU, GLU variants have better smoothness and similar convergence speed. BCAT is the first to test and demonstrate their effectiveness in scientific computing. (4) BCAT shows the benefit of causality in terms of efficiency and performance.

While models like MPP and DPOT are designed to be fixed window update rules, causality can also be integrated to improve their performance, similar to the Time-then-space variant of BCAT.

Better engineering designs. With the goal of building large-scale PDE foundation models, we take engineering aspects into consideration. The backbone of BCAT is a standard decoder-only transformer. While we change the masking and the learning objective, the main architecture is not altered. This comes with several benefits. First, this is the only architecture that shows scaling ability beyond 10 billion parameters. Larger models trained using AFNO (i.e., DPOT) or axial attention (MPP) do not exist yet. Second, many recent engineering improvements are designed only for standard structures, not for certain custom operations. For instance, BCAT is compatible with improvements such as `bfloat16` mixed precision, `torch.compile` for kernel fusion, and `FlexAttention` [DFG+24] to utilize sparsity in block causal mask, which are not compatible with MPP or DPOT. As a reference, on the same machine for similar-sized models, the training time is 42 hours for BCAT, 120 hours for MPP-B, and 80 hours for DPOT-M.

4.5 Theoretical Analysis of BCAT

In this section, we provide a theoretical analysis of the BCAT architecture, with the goal of demonstrating the universality of BCAT as an approximator for spatiotemporal dynamics.

We first establish the necessary mathematical framework for our analysis, including notations and definitions of appropriate function spaces. We then present our main theoretical result, which states that BCAT is a universal approximator for continuous (deterministic) Markovian functions.

4.5.1 Notations and Preliminary Results

Index and Token Domain. Let the discretized spatiotemporal sequence be a compact hypercube $\mathcal{K} \subseteq [0,1]^{d\times N}$, where d is the embedding dimension and $N = T \cdot S$ is the total sequence length (T time steps, S spatial modes). We index a sequence $X \in \mathcal{K}$ by its time step $t \in [T] := \{1, \dots, T\}$ and spatial mode $s \in [S]$. The continuous latent token at (t, s) is $x_{t,s} \in [0,1]^d$. We denote the whole sequence as $X = \{x_{t,s}\}_{t,s}$.

Function Spaces. Let $\mathcal{F}_{bc}$ be the space of continuous, block-causal functions $F : \mathcal{K} \to \mathbb{R}^{d\times N}$. Block-causality dictates that the output at time t depends strictly on the history up to time t. Formally, for any $X, \tilde{X} \in \mathcal{K}$, if $x_{t',s} = \tilde{x}_{t',s}$ for all $t' \le t$ and $s \in [S]$, then $F(X)_{t,s} = F(\tilde{X})_{t,s}$ for all $s \in [S]$.

Block Causal Mask (M). The mask $M \in \{0, -\infty\}^{N\times N}$ is applied pre-softmax. For a query at (t, s) and key at (τ, σ),

$$M_{(t,s),(\tau,\sigma)} = \begin{cases} 0 & \text{if } \tau \le t \\ -\infty & \text{otherwise} \end{cases} \tag{4.4}$$

Note that, we use an additive $0/-\infty$ mask here for theoretical analysis, which is equivalent to the on/off boolean mask (4.2).

Network Architecture. BCAT transformer consists of multiple layers of masked multi-head self attention (MHA) and feedforward networks (FFN). Both layers contains residual connections and layer normalization. We omit the layer normalization for our analysis. The MHA layer with H heads and head dimension d_h is defined as follows. For input sequence $X \in \mathbb{R}^{d\times N}$, the output at head $h \in [H]$ is

$$\text{head}_h(X) = \text{softmax}\left((W_h^Q X + b_h^Q)^\top (W_h^K X + b_h^K) + M\right)(W_h^V X + b_h^V), \tag{4.5}$$

where $W_h^Q, W_h^K, W_h^V \in \mathbb{R}^{d_h\times d}, b_h^Q, b_h^K, b_h^V \in \mathbb{R}^{d_h}$ are the query, key, and value projection weights and bias for head h. The MHA output is the concatenation of all heads, i.e.

$$\text{MHA}(X) = \text{concat}(\text{head}_1(X), \dots, \text{head}_H(X)). \tag{4.6}$$

The FFN layer is defined as $\text{FFN}(X) = W_2\sigma(W_1X)$, where $W_1 \in \mathbb{R}^{d_{ff}\times d}$ and $W_2 \in \mathbb{R}^{d\times d_{ff}}$ are the projection matrices and σ is a non-linear activation function (e.g. ReLU).

In the following analysis, we will make use of standard universal approximation results for MLPs, so we include the statement for reference.

Theorem 4.5.1 (Universal Approximation of MLPs [KL20]). *Let $\rho : \mathbb{R} \to \mathbb{R}$ be a non-affine continuous function which is continuously differentiable at at least one point, with non-zero derivative at that point. Let $\mathcal{NN}^{\rho}_{n,m,k}$ represent the class of functions $\mathbb{R}^n \to \mathbb{R}^m$ described by feedforward neural networks with n neurons in the input layer, m neurons in the output layer, and an arbitrary number of hidden layers, each with k neurons with activation function ρ. Let $K \subseteq \mathbb{R}^n$ be compact, then $\mathcal{NN}^{\rho}_{n,m,n+m+2}$ is dense in $C(K;\mathbb{R}^m)$ with respect to the uniform norm.*

4.5.2 Overview

Before proceeding to the lemmas and theorems, we outline the high-level mechanics of the proof. The goal is to demonstrate that a block-causal attention transformer (BCAT) can act as a universal approximator for the discretized solution operator for autonomous dynamical systems.

We first establish that BCAT does not need to learn arbitrary sequence mappings, but rather a specific class of functions. Using the Mori-Zwanzig formalism [Zwa61, Mor65], we demonstrate that the non-Markovian solution operator of a projected dynamical system can be represented as a generalized Langevin equation. The resulting numerical flow map belongs to the class of continuous block-causal functions ($\mathcal{F}_{bc}$). Therefore, it is sufficient to show that BCAT approximate any continuous block-causal function.

To prove that the BCAT approximates the block-causal class $\mathcal{F}_{bc}$, we construct the network to process the spatiotemporal sequence in substeps:

1. Discretization of the Input via FFN (Lemma 4.5.2): We process the continuous input

sequence through an initial element-wise FeedForward Network. This network acts as a continuous step-function approximator, mapping continuous variables to a finite, uniformly spaced spatial grid. This effectively reduces the continuous state space into a finite set of discrete possible sequence histories.

2. Separation of Context via Attention (Lemma 4.5.3): We configure the masked multi-head attention layer to act as a flawless memory buffer. In an ideal setting, we let attention layer perform a "history lookup," generating a concatenated vector that isolates and uniquely identifies the exact valid history of the system up to the current time step. Then we show that difference between the ideal output and actual output can be arbitrarily small.

3. Mapping Outputs Back to the State Space (Lemma 4.5.4): As Step 2 guarantees that every unique sequence history is separated into a distinct context vector, we can then invoke the standard Universal Approximation Theorem, using a final element-wise FFN to map these isolated context vectors to the exact continuous target values of the underlying block-causal solution operator.

By proving that the approximation errors at each of these three substeps can be strictly bounded, we synthesize the final theorem demonstrating the universality of BCAT.

4.5.3 Lemmata

beginsep=**Lemma 4.5.2** (Input Quantization)**.** *For any $\delta > 0$ and $0 < \varepsilon_1 < 1$, there exists a FeedForward Network (FFN) $\Phi_{quant} : \mathbb{R}^d \to \mathbb{R}^d$ s.t. for any $X \in \mathcal{K}$, the output sequence $\bar{X}$ (where $\bar{x}_{t,s} = \Phi_{quant}(x_{t,s})$ element-wise) belongs to a finite discrete set of grid points $\mathcal{K}_\delta \subseteq \mathcal{K}$. Furthermore, there exists a "well-quantized" subset $U \subseteq \mathcal{K}$ s.t. the Lebesgue measure of the complement $\mu(\mathcal{K} \setminus U) < \varepsilon_1$ and for all $X \in U$, $\left\|X - \bar{X}\right\|_\infty < \delta$.*

beginsep=6plus6We partition $[0, 1]^d$ into a grid of hypercubes with side length δ. We consider the function f mapping any point within each subgrid to its center, which is a

piecewise constant function thus measurable. By Lusin's theorem, we can find a continuous function g and a closed subset $U \subset \mathcal{K}$ s.t. $f = g$ on U, and $\mu(\mathcal{K} \setminus U) < \varepsilon_1$. Then by standard universal approximation results for MLPs (Theorem 4.5.1), there exists a FFN Φ_{quant} that approximates g to arbitrary precision, thus also approximating f on U. The output of Φ_{quant} is then a quantized version of the input, and the approximation error can be made smaller than δ for all inputs in U. □

*Proof. beginsep=***Lemma 4.5.3** (Contextual Separation)**.** *Let $\mathcal{K}_\delta$ be the finite set of quantized sequences. For any error tolerance $\gamma > 0$, there exists an absolute positional encoding strategy and a block-causal multi-head self-attention (MHA) layer utilizing mask M s.t. the generated output embeddings $H = L(X) \in \mathbb{R}^{d \times N}$ uniquely identify valid block-causal histories. Here L represents the MHA layer with the chosen positional encoding and residual connection. More precisely, if two sequences $X, Y \in \mathcal{K}_\delta$ have distinct histories up to time t ($X_{\leq t} \neq Y_{\leq t}$), then there exists an ideal mapping L^{ideal} s.t. $L^{ideal}(X)_{t,s} \neq L^{ideal}(Y)_{t,s}$, and the actual learnable mapping L satisfies $\left\|L^{ideal}(X) - L(X)\right\|_2 < \gamma$.*

beginsep=6plus6We denote the $\bar{X}$ as the quantized version of X obtained from Lemma 4.5.2.

Proof. **Step 1: Subspace Separation.** To simplify the orthogonality design, we embed data and positional embeddings into a higher-dimensional space $\mathbb{R}^{d+N}$. Here

1. Data: $\tilde{x}_{t,s} = \begin{bmatrix} x_{t,s} \\ \mathbf{0}_N \end{bmatrix} \in \mathbb{R}^{d+N}$
2. Positional embedding: $e_{t,s} = \begin{bmatrix} \mathbf{0}_d \\ v_{t,s} \end{bmatrix} \in \mathbb{R}^{d+N}$, where $\{v_{t,s}\}$ is a set of orthonormal vectors in $\mathbb{R}^N$.
3. Input: the input vector is $z_{t,x} = \tilde{x}_{t,s} + e_{t,s} \in \mathbb{R}^{d+N}$.

Here we do a dimension lifting to decouple the data and the positional information, which allows us to design the attention weights to achieve perfect history separation. Usually, transformers have a fixed embedding dimension ($d_{in} = d_{out}$), but this can be fixed by merging

the extra uplifting projection into the first layer of the network. In practice, when d is large enough, we can find sufficient amount of (nearly) orthogonal directions in $\mathbb{R}^d$, eliminating the need for dimension lifting.

Step 2: Construct Attention Matrices. We assume the MHA layer has a total of $H = N$ heads, each corresponding to one space-time coordinate (τ, σ). For head $(\tau, \sigma) \in [T] \times [S]$, we define the query, key, value projection weights and bias as follows:

1. Query: $W^Q_{\tau,\sigma} = \mathbf{0}_{(d+N)\times(d+N)}$, $b^Q_{\tau,\sigma} = \alpha \cdot e_{\tau,\sigma}$
2. Key: $W^K_{\tau,\sigma} = I_{(d+N)\times(d+N)}$, $b^K_{\tau,\sigma} = \mathbf{0}_{d+N}$
3. Value: $W^V_{\tau,\sigma} = \begin{bmatrix} I_{d\times d} & \mathbf{0}_{d\times N} \\ \mathbf{0}_{N\times d} & \mathbf{0}_{N\times N} \end{bmatrix}$, $b^V_{\tau,\sigma} = \mathbf{0}_{d+N}$

With these projection layers, the pre-softmax attention score for head (τ, σ), query location (t, s) and key location (τ', σ') (i.e., the $((t, s), (\tau', \sigma'))$ entry for head (τ, σ)) is given by

$$\begin{aligned} A_{(t,s),(\tau',\sigma')} &= \left((W^Q_{\tau,\sigma} Z + b^Q_{\tau,\sigma})^T (W^K_{\tau,\sigma} Z + b^K_{\tau,\sigma}) + M\right)_{(t,s),(\tau',\sigma')} \\ &= \alpha \cdot e^T_{\tau,\sigma} e_{\tau',\sigma'} + M_{(t,s),(\tau',\sigma')} \\ &= \alpha \cdot \delta_{(\tau,\sigma),(\tau',\sigma')} + M_{(t,s),(\tau',\sigma')} \end{aligned}$$

where δ is the Kronecker delta function. If $\tau > t$, the block causal mask enforces the attention score to be $-\infty$, and the head output after softmax is $\mathbf{0}_{d+N}$. If $\tau \leq t$, the mask M is 0, and score is α in column (τ, σ) and 0 in all other columns.

Step 3: Ideal argmax attention case. If softmax in attention is replaced by argmax, the ideal output of head (τ, σ) is

$$h^{ideal}_{\tau,\sigma} := \text{head}^{ideal}_{\tau,\sigma}(Z) = \text{argmax}\left((W^Q_{\tau,\sigma} Z + b^Q_{\tau,\sigma})^\top (W^K_{\tau,\sigma} Z + b^K_{\tau,\sigma}) + M\right)(W^V_{\tau,\sigma} Z + b^V_{\tau,\sigma}) \quad (4.7)$$

$$= \text{argmax}(A)\tilde{X} \in \mathbb{R}^{N\times(d+N)} \quad (4.8)$$

and the (t, s) entry of $h^{ideal}_{\tau,\sigma}$ is

$$(h^{ideal}_{\tau,\sigma})_{t,s} = \begin{cases} \tilde{x}_{\tau,\sigma} & \text{if } \tau \leq t \text{ (key is unmasked)} \\ \frac{1}{k}\sum_{\tau'=1}^{t}\sum_{\sigma'=1}^{S}\tilde{x}_{\tau',\sigma'} & \text{o.w. (key is masked, argmax is average of } k \text{ valid tokens)} \end{cases} \tag{4.9}$$

Here we assume this row has a total of k non-infinity scores. In this case, we define

$$L^{ideal}(X)_{t,s} = \begin{bmatrix} (h^{ideal}_{1,1})_{t,s} \\ \cdots \\ (h^{ideal}_{T,S})_{t,s} \\ Z \end{bmatrix} \tag{4.10}$$

and consider two sequences $X, Y \in \mathcal{K}_\delta$. If the histories differ ($X_{\leq t} \neq Y_{\leq t}$), then the historical summaries (output from attention head) differ, as they are concatenations of all unmasked history tokens. If histories are the same but spatial coordinates differ (same t but different s), then the positional encoding from residual connection will differ. In either case, we have $L^{ideal}(X)_{t,s} \neq L^{ideal}(Y)_{t,s}$, which means the output embedding at (t, s) can uniquely identify the history up to time t. Here we use a similar trick as in step 1 to do dimension lifting, which turns residual addition into concatenation.

Step 4: Softmax attention case. Now for the actual output with softmax, we assume there are $k - 1$ entries with score 0, i.e., this row has a total of k non-infinity scores. Then the output of head (τ, σ) is given by

$$(h_{\tau,\sigma})_{t,s} = \begin{cases} \frac{e^\alpha}{e^\alpha+k-1}\tilde{x}_{\tau,\sigma} + \sum_{other}\frac{1}{e^\alpha+k-1}\tilde{x}_{other} & \text{if } \tau \leq t \text{ (key is unmasked)} \\ \frac{1}{k}\sum_{\tau'=1}^{t}\sum_{\sigma'=1}^{S}\tilde{x}_{\tau',\sigma'} & \text{otherwise (key is masked)} \end{cases} \tag{4.11}$$

We now bound the distance between the actual softmax output and the ideal argmax output. For heads where $t < \tau$, the outputs are the same thus error is 0. For heads where $t \geq \tau$, the

error is caused by softmax leakage:

$$\left\| \frac{e^\alpha}{e^\alpha + k - 1} \tilde{x}_{\tau,\sigma} + \sum_{other} \frac{1}{e^\alpha + k - 1} \tilde{x}_{other} - \tilde{x}_{\tau,\sigma} \right\|_2 \tag{4.12}$$

$$= \left\| \sum_{other} \frac{1}{e^\alpha + k - 1} (\tilde{x}_{other} - \tilde{x}_{\tau,\sigma}) \right\|_2 \tag{4.13}$$

$$\leq \sum_{other} \frac{1}{e^\alpha + k - 1} \left\| \tilde{x}_{other} - \tilde{x}_{\tau,\sigma} \right\|_2 \tag{4.14}$$

$$\leq \frac{k-1}{e^\alpha + k - 1} \cdot \mathrm{diam}(\mathcal{K}) \tag{4.15}$$

$$\leq \frac{N}{e^\alpha} \sqrt{d}. \tag{4.16}$$

Summing this error for all heads give us the total ℓ^2 error bound:

$$\sqrt{N \frac{N^2}{e^{2\alpha}} d} = \frac{N^{1.5}\sqrt{d}}{e^\alpha} < \gamma \;\Leftrightarrow\; \alpha > \log\left(\frac{N^{1.5}\sqrt{d}}{\gamma} \right). \tag{4.17}$$

Thus if we choose α large enough, the output of the MHA layer can be made arbitrarily close to the ideal output, which has perfect history separation property. □

beginsep=**Lemma 4.5.4** (Output Mapping)**.** *Given the ideal context embeddings $L^{ideal}(\bar{X})$ (which uniquely identify valid histories and spatial coordinates), and error tolerance $\varepsilon_2 > 0$, there exists a FeedFoward Network Φ_{out} and a output tolerance $\gamma > 0$ s.t., for any generates context embedding $L(\bar{X})$ satisfying $\left\| L(\bar{X}) - L^{ideal}(\bar{X}) \right\|_2 < \gamma$, the output satisfies*

$$\left\| \Phi_{out}(L(\bar{X})) - F(X) \right\|_2 < \varepsilon_2. \tag{4.18}$$

beginsep=6plus6By Lemma 4.5.3, the set of ideal context outputs $\{L^{ideal}(\bar{X})_{t,s} \mid \bar{X} \in \mathcal{K}_\delta, t \in [T], s \in [S]\}$ is finite. Furthermore, the residual positional embeddings ensures that each ideal context output vector is a bijection to the tuple $(\bar{X}_{\leq t}, s)$.

Proof. We define a function g over this finite set of ideal vectors, $g(L^{ideal}(\bar{X})_{t,s}) = F(\bar{X})_{t,s}$. As $F \in \mathcal{F}_{bc}$ is block-causal, $F(\bar{X})_{t,s}$ depends on $\bar{X}_{\leq t}$ and s. The bijectivity of ideal context vectors ensures that g is a well-defined mapping from a finite set of discrete points to $\mathbb{R}^d$.

Therefore, by universal approximation of MLPs (Theorem 4.5.1), there exists a FFN (with ReLU activation) Φ_{out} that interpolates these exact points: $\Phi_{out}(L^{ideal}(\bar{X})_{t,s}) = F(\bar{X})_{t,s}$. Additionally, since Φ_{out} is continuous function on a bounded domain, it is uniformly continuous, and we can find $\gamma > 0$ s.t. $\|y - y'\|_2 < \gamma \;\Rightarrow\; \|\Phi_{out}(y) - \Phi_{out}(y')\|_2 < \varepsilon_2$.

Now apply Lemma 4.5.3 with this choice of γ, i.e., we choose $\alpha > \log\left(N^{1.5}\sqrt{d}\gamma^{-1}\right)$, we ensure the actual attention output $L(\bar{X})$ is γ-close to the ideal output $L^{ideal}(\bar{X})$, and

$$\left\|\Phi_{out}(L(\bar{X})) - F(X)\right\|_2 < \varepsilon_2. \tag{4.19}$$

□

4.5.4 Main Approximation Theorem

We are now ready to present the main theorem, utilizing the above lemmata to construct a BCAT network that approximates any continuous block-causal function to arbitrary precision. Additionally, we show that this is a practical function space, as it includes the solution operator for any continuous autonomous dynamical system (e.g. Navier-Stokes simulations). While the discretization of the dynamical system may break the Markovian property, the Mori-Zwanzig formalism suggests that block-causal functions can approximate the discretized dynamics.

*beginsep=***Theorem 4.5.5** (Universal Approximator of Block-Causal Functions)**.** *For $1 \leq p < \infty$ and any continuous block-causal function $F \in \mathcal{F}_{bc}$, any compact domain of spatiotemporal sequences $\mathcal{K} \subseteq [0,1]^{d\times N}$, and error tolerance $\varepsilon > 0$, there exists a block-causal Transformer network T_θ defined by:*

$$T_\theta(X) = \Phi_{out}\left(Concat[MHA(\Phi_{quant}(X) + E_{pos}), \Phi_{quant}(X) + E_{pos}]\right) \tag{4.20}$$

such that

$$\|T_\theta - F\|_{L^p(\mathcal{K})} < \varepsilon. \tag{4.21}$$

beginsep=6plus6As F is continuous on a compact domain $\mathcal{K}$, it is uniformly continuous and bounded by some scalar $M_F > 0$. BY uniform continuity, there exists a quantization resolution $\delta > 0$ s.t.

$$X, Y \in \mathcal{K} \text{ and } \|X - Y\|_\infty < \delta \;\Rightarrow\; \|F(X) - F(Y)\|_p < \frac{\varepsilon}{3}. \tag{4.22}$$

Proof. We now construct the BCAT network using previous lemmas.

1. Input Quantization: We apply Lemma 4.5.2 and obtain an FFN Φ_{quant} with this choice of δ, and complement $(\mathcal{K} \setminus U)$ measure bound $\varepsilon_1 < \left(\frac{\varepsilon}{6M_F}\right)^p$.

2. Target Mapping: We apply Lemma 4.5.4 to construct the final MLP Φ_{out} with error tolerance $\varepsilon_2 = \varepsilon/3$, which yields a softmax attention leakage tolerance γ. Note that Φ_{out} is constructed based on the ideal argmax attention output, which does not depend on the choice of α and γ.

3. Context Separation: We apply Lemma 4.5.3 to construct the block-causal attention layer, setting the attention scale $\alpha > \log(N^{1.5}\sqrt{d}\gamma^{-1})$.

Now for the final error bound, we split the error into network error and quantization error:

$$\|T_\theta - F\|_{L^p(\mathcal{K})} \le \left\|T_\theta(X) - F(\bar{X})\right\|_{L^p(\mathcal{K})} + \left\|F(\bar{X}) - F(X)\right\|_{L^p(\mathcal{K})}. \tag{4.23}$$

For the network error, by Lemma 4.5.3 and 4.5.4, for any $\bar{X} \in \mathcal{K}_\delta$, the pointwise error is bounded by $\varepsilon_2 = \varepsilon/3$, and thus

$$\left\|T_\theta(X) - F(\bar{X})\right\|_{L^p(\mathcal{K})} \le \mu(\mathcal{K}) \cdot \varepsilon/3 \le \varepsilon/3. \tag{4.24}$$

For the quantization error, we decompose the domain into two regions, U and $\mathcal{K} \setminus U$. On U, input error is less than δ, thus uniform continuity of F ensures the output error is less

than $\varepsilon/3$. On $\mathcal{K} \setminus U$, the worst-case bound is $2M_F$. Therefore,

$$\left\|F(X) - F(\bar{X})\right\|_{L^p(\mathcal{K})} = \left(\int_U \left\|F(\bar{X}) - F(X)\right\|_p^p \, d\mu + \int_{\mathcal{K}\setminus U} \left\|F(\bar{X}) - F(X)\right\|_p^p \, d\mu\right)^{1/p} \tag{4.25}$$

$$\leq \left(\mu(U) \cdot \frac{\varepsilon^p}{3^p} + \mu(\mathcal{K} \setminus U) \cdot (2M_F)^p\right)^{1/p} \tag{4.26}$$

$$\leq \left(1 \cdot \frac{\varepsilon^p}{3^p} + \varepsilon_1 \cdot (2M_F)^p\right)^{1/p} \tag{4.27}$$

$$\leq \frac{2\epsilon}{3}. \tag{4.28}$$

Combining the two error bounds yields the final result:

$$\|T_\theta - F\|_{L^p(\mathcal{K})} \leq \left\|T_\theta(X) - F(\bar{X})\right\|_{L^p(\mathcal{K})} + \left\|F(\bar{X}) - F(X)\right\|_{L^p(\mathcal{K})} \leq \frac{\varepsilon}{3} + \frac{2\varepsilon}{3} = \varepsilon. \tag{4.29}$$

□

4.5.5 Approximation of Dynamical System Solution Operators

We now show that the function space of block-causal functions is rich enough to include the solution operator for continuous autonomous dynamical systems. We first introduce the Mori-Zwanzig formalism [Zwa61, Mor65].

The Mori-Zwanzig formalism provides a framework for deriving reduced models for high-dimension dynamical systems. Given state variable $\Gamma(t)$ evolving according to the dynamical system $\dot{\Gamma}(t) = \mathcal{L}\Gamma(t)$, where $\mathcal{L}$ is the Liouville operator. Let $\mathcal{P}$ be a linear projection operator from the full state space to a subspace of interest (e.g. discretization), with projected variable $A(t) := \mathcal{P}\Gamma(t)$, the Mori-Zwanzig theorem states the the evolution of the projected variable can be described by a generalized Langevin equation:

$$\dot{A}(t) = \mathcal{P}\mathcal{L}A(t) + \int_0^t K(t-s)A(s) \, ds + F(t), \tag{4.30}$$

where:

1. $\mathcal{P}\mathcal{L}A(t)$ captures instantaneous Markovian dynamics,

2. $\int_0^t K(t-s)A(s)\ ds$ represents the memory effects, encoding history dependence, and $K(t) = \mathcal{P}\mathcal{L}e^{t(1-\mathcal{P})\mathcal{L}}(1-\mathcal{P})\mathcal{L}$ is the memory kernel.

3. $F(t) = \mathcal{P}\mathcal{L}e^{t(1-\mathcal{P})\mathcal{L}}(1-\mathcal{P})\Gamma(0)$ is the random force, a noise term arising from degrees of freedom in the $(1-\mathcal{P})$ subspace.

Block-causal functions explicitly capture the memory term induced by subspace projection to approximate the original dynamical system.

beginsep=**Corollary 4.5.6** (Universal Approximator of Autonomous Dynamical Systems)**.** *Let the groundtruth autonomous dynamical system be defined on a state space $\mathcal{H}$, governed by*

$$\dot{u}(t) = \mathcal{L}u(t). \tag{4.31}$$

Let $\mathcal{P} : \mathcal{H} \to \mathbb{R}^{d\times S}$ be a spatial projection operator (e.g., discretization, autoencoder latent projection) that maps the full state to a resolved, lower-dimensional observable $x(t) = \mathcal{P}u(t)$. We sample this continuous observable over a uniform temporal grid with step size Δt to generate a discrete sequence of length T. The complete spatiotemporal sequence is denoted as $X = [x_1, \ldots, x_T] \in \mathbb{R}^{d\times N}$, where $N = T \cdot S$ is the total sequence length. Let $\mathcal{K} \subseteq [0,1]^{d\times N}$ be a compact domain of projected spatiotemporal sequences, and let $\mathcal{F}_{proj} : \mathcal{K} \to \mathcal{K}$ be the evolution operator of the projected system, s.t.

$$\mathcal{F}_{proj}(x_1, \ldots, x_T) = (x_2, \ldots, x_{T+1}). \tag{4.32}$$

Assuming the unresolved initial state $(I-\mathcal{P})u(0)$ is a continuous function of the resolved initial state $\mathcal{P}u(0)$, for $1 \le p < \infty$ and any $\varepsilon > 0$, there exists a BCAT network T_θ such that

$$\|T_\theta - \mathcal{F}_{proj}\|_{L^p(\mathcal{K})} < \varepsilon. \tag{4.33}$$

beginsep=6plus6We use the Mori-Zwanzig formalism to connect the resolved evolution operator and the block-causal function class. By applying the Mori-Zwanzig theorem to the resolved observable $x(t) = \mathcal{P}u(t)$, we have the following evolution equation:

$$\dot{x}(t) = \mathcal{P}\mathcal{L}x(t) + \int_0^t K(t-s)x(s)\, ds + F(t). \tag{4.34}$$

Proof. We discretize this integro-differential equation over the temporal grid with step size Δt:

$$x_{t+1} \approx \mathcal{G}_{MZ}(x_0, \ldots, x_t) := x_t + \Delta t \cdot \left(\mathcal{P}\mathcal{L}x(t) + \sum_{s=0}^{t} w_s K(t-s)x(s) + F(t) \right), \tag{4.35}$$

where w_s are the quadrature weights for approximating the integral. This discretized evolution equation is a block-causal function, as the next state x_{t+1} depends on the history of states $x_{\leq t}$ through the memory term. Note that our assumption on the unresolved initial state ensures the random force term $F(t)$ is also a continuous function of the resolved initial state, thus can be absorbed into the block-causal function. To summarize, the solution operator for the discretized system can be approximated by a block-causal function $\mathcal{G}_{MZ}$, which maps the history of states to the next state. By choosing proper $\Delta > 0$ and quadrature rule, we can make the discretization error $\|\mathcal{G}_{MZ} - \mathcal{F}_{proj}\|_{L^p(\mathcal{K})} < \varepsilon/2$.

Now as $\mathcal{G}_{MZ} \in \mathcal{F}_{bc}$ is a continuous block-causal function, we invoke Theorem 4.5.5 to construct a BCAT network T_θ that approximates $\mathcal{G}_{MZ}$ to arbitrary precision, i.e., $\|T_\theta - \mathcal{G}_{MZ}\|_{L^p(\mathcal{K})} < \varepsilon/2$. Combining the two error bounds yields the final result. □

4.6 Conclusion

In this work, we introduced BCAT, a novel PDE foundation model designed for autoregressive prediction of fluid dynamics solutions. Leveraging a block causal transformer architecture, BCAT addresses the limitations of existing models by focusing on next frame prediction rather than next token prediction. This design captures the complex spatiotemporal dependencies inherent in fluid dynamics more effectively, leading to improved accuracy

and computational efficiency. Our experimental results highlight the strengths of BCAT in various popular benchmark datasets, including PDEBench, PDEArena, and CFDBench. BCAT consistently outperformed state-of-the-art models across diverse evaluation tasks, in both zero-shot predictions and fine-tuning experiments. Notably, BCAT demonstrated superior parameter efficiency, outperforming larger models with 3.4 times more parameters, and achieved notable improvements over similarly sized models. The ablation study further validated the advantages of next frame prediction approach, which enhanced accuracy and efficiency in modeling nonlinear physical phenomena in fluid dynamics. This finding underscores the effectiveness of the block causal framework in capturing complex dependencies. BCAT is the first to apply the Muon optimizer for scientific computing and prediction tasks, which we found better captures high contrast regions compared to AdamW and significantly improves the performance. BCAT represents a significant step forward in the development of PDE foundation models, providing a powerful and efficient tool for advancing scientific computing and fluid dynamics research.

4.A Dataset Details

The data was obtained from the PDEBench [TPL+22], PDEArena [GB22], and CFDBench [LCZ23] datasets. Unless otherwise specified, the space resolution is 128×128.

4.A.1 PDEBench

Shallow Water Equation. The quantity of interest is the water depth $h(\boldsymbol{x}, t)$ on domain $[-2.5, 2.5]^2 \times [0, 1]$ with Neumann boundary condition. The temporal resolution is 101. The equations are:

$$\partial_t h + \nabla h\boldsymbol{u} = 0, \tag{4.36}$$

$$\partial_t h\boldsymbol{u} + \nabla \left(h\,\boldsymbol{u} \cdot \boldsymbol{u} + \frac{1}{2} g_r h^2 \right) = -g_r h \nabla b. \tag{4.37}$$

Incompressible Navier-Stokes Equation. The quantities of interest are the velocities $\boldsymbol{u}(\boldsymbol{x}, t)$ and particle density $c(\boldsymbol{x}, t)$ on domain $[0, 1]^2 \times [0, 5]$ with Dirichlet boundary condition. The temporal resolution is 1000. The equations are:

$$\rho(\partial_t \boldsymbol{u} + \boldsymbol{u} \cdot \nabla \boldsymbol{u}) = -\nabla p + \mu \Delta \boldsymbol{u} + \mathbf{F}, \tag{4.38}$$

$$\nabla \cdot \boldsymbol{u} = 0, \tag{4.39}$$

$$\partial_t c + \nabla \cdot (c\boldsymbol{u}) = 0. \tag{4.40}$$

The forcing term $\mathbf{F}$ is randomly sampled.

Compressible Navier-Stokes Equation. The quantities of interest are the velocities $\boldsymbol{u}(\boldsymbol{x}, t)$, pressure $p(\boldsymbol{x}, t)$, and density $\rho(\boldsymbol{x}, t)$ on domain $[0, 1]^2 \times [0, 1]$ with periodic boundary conditions. The temporal resolution is 21. For equations with low viscosities, the dataset is provided on a finer 512×512 space grid, which is downsampled to 128×128 for consistency (through average pooling). The equations are:

$$\partial_t \rho + \nabla \cdot (\rho \boldsymbol{u}) = 0, \tag{4.41}$$

$$\rho(\partial_t \boldsymbol{u} + \boldsymbol{u} \cdot \nabla \boldsymbol{u}) = -\nabla p + \eta \Delta \boldsymbol{u} + (\zeta + \eta/3) \nabla (\nabla \cdot \boldsymbol{u}), \tag{4.42}$$

$$\partial_t \left(\varepsilon + \frac{\rho u^2}{2} \right) = -\nabla \cdot \left(\left(\varepsilon + p + \frac{\rho u^2}{2} \right) \boldsymbol{u} - \boldsymbol{u} \cdot \sigma' \right). \tag{4.43}$$

4.A.2 PDEArena

Incompressible Navier-Stokes Equation. The quantities of interest are the velocities $\boldsymbol{u}(\boldsymbol{x}, t)$ and particle density $c(\boldsymbol{x}, t)$ on domain $[0, 32]^2 \times [18, 102]$ with Dirichlet boundary conditions for velocity and Neumann boundary condition for particle field. The temporal resolution is 14. The equations are:

$$\rho(\partial_t \boldsymbol{u} + \boldsymbol{u} \cdot \nabla \boldsymbol{u}) = -\nabla p + \mu \Delta \boldsymbol{u} + \mathbf{F}, \tag{4.44}$$

$$\nabla \cdot \boldsymbol{u} = 0, \tag{4.45}$$

$$\partial_t c + \nabla \cdot (c\boldsymbol{u}) = 0. \tag{4.46}$$

The forcing term $\mathbf{F}$ takes the form $\mathbf{F} = (0, f)$ with $f = 0.5$.

Incompressible Navier-Stokes Equation (Conditioned). The quantities of interest are the velocities $\boldsymbol{u}(\boldsymbol{x}, t)$ and particle density $c(\boldsymbol{x}, t)$ on domain $[0, 32]^2 \times [18, 102]$ with Dirichlet boundary conditions for velocity and Neumann boundary condition for particle field. The temporal resolution is 56. The equations are:

$$\rho(\partial_t \boldsymbol{u} + \boldsymbol{u} \cdot \nabla \boldsymbol{u}) = -\nabla p + \mu \Delta \boldsymbol{u} + \mathbf{F}, \tag{4.47}$$

$$\nabla \cdot \boldsymbol{u} = 0, \tag{4.48}$$

$$\partial_t c + \nabla \cdot (c\boldsymbol{u}) = 0. \tag{4.49}$$

The forcing term $\mathbf{F}$ takes the form $\mathbf{F} = (0, f)$ where f is uniformly sampled in $[0.2, 0.5]$.

4.A.3 CFDBench

Incompressible Navier-Stokes Equation. The quantities of interest are the velocities $\boldsymbol{u}(\boldsymbol{x}, t)$ and pressure $p(\boldsymbol{x}, t)$. This dataset contains irregular geometries with Dirichlet boundary conditions. The raw space resolution is 64×64 which is upsampled to 128×128 via interpolation. The equations are:

$$\rho(\partial_t \boldsymbol{u} + \boldsymbol{u} \cdot \nabla \boldsymbol{u}) = -\nabla p + \mu \Delta \boldsymbol{u}, \tag{4.50}$$

$$\nabla \cdot \boldsymbol{u} = 0. \tag{4.51}$$

4.B Experiment Details

4.B.1 Training

We perform data normalization during the training process. Given the input sequence of data $\{\boldsymbol{u}(\cdot, t_i) \mid 0 \leq i < T_0\}$, we compute the mean and standard deviation of each input trajectory, which are used to normalize both the input and ground truth sequence. The loss function is the standard mean squared error in the normalized space. The models are trained using

a combination of Muon optimizer [JJB+24, LSY+25] and AdamW optimizer [LH17] with a global batch size of 128 for 40 epochs where each epoch is 4,000 steps. The Muon optimizer is applied to all transformer matrix parameters, while AdamW optimizer is applied to the rest of the parameters (scalar, embedding, initial and final projection layer). The warmup-stable-decay learning rate scheduler [HTH+24] is used with 10% warmup and 50% decay. We use learning rate 10^{-3} and weight decay 10^{-4}. On four NVIDIA H100 GPUs, the training takes about 42 hours.

4.B.2 Model Hyperparameters

We summarize the model hyperparameters in Table 4.9. An illustration of a transformer layer used in BCAT model is shown in Figure 4.4.

Table 4.9: **Model hyperparameters.** FFN means feedforward network.

Hidden dimension - attention	1024	Hidden dimension - FFN	2752
Number of attention heads	8	Number of layers	12
Activation	SwiGLU	Dropout	0
Patch Size	8		

4.B.3 Baselines and Comparisons

In this section, we include more details about the compared models.

DeepONet [LJK19]. We employ the unstacked DeepONet architecture, consisting of a single trunk network and a single branch network. Initially, the input data is divided into 8×8 patches, with each patch being embedded into a 128-dimensional vector. These vectors are then passed through the branch network, producing an output with a basis dimension of $p = 50$. Simultaneously, the query point is processed through the trunk network, which also outputs a vector with the same dimension, p. The output solution at the query point is

obtained by taking the inner product of the outputs from the two networks.

FNO [LKA+21]. We use 4 layers of standard 3d FNO to process the input data. The number of modes to keep in each dimension is set to 8, and the number of hidden channels is set to 16. The 3d FNO model directly maps 10 input steps to 10 output steps.

UNet [RFB15]. We use 8 layers of 3d UNet with GeLU activation and 32 hidden dimensions. The 3d UNet model directly maps 10 input steps to 10 output steps (Padding is added to the input to make it a power of 2, which is needed for the shrinkage in dimension).

ViT [DBK+21]. For ViT, we use 12 transformer layers similar to the one shown in Figure 4.4. The patch size is set to be 16, the hidden dimension for attention is 1024, the hidden dimension for the feedforward network is 4096 with GeLU activation, and the number of heads is 8. The ViT model learns a fixed window update rule, directly mapping 10 input steps to 10 output steps.

PROSE-FD [LSH+24]. We directly use the results for PROSE-FD in [LSH+24], since the same dataset and same context window is used.

MPP [MRSBP+23]. In [MRSBP+23], MPP is trained on a different training dataset and different evaluation metrics are used. For a fair comparison, we retrained MPP-B and MPP-L on our training dataset keeping the same model configurations and optimizer hyperparameters. We evaluate MPP-B and MPP-L using the same testing setup and metric, where rollout is used to obtain all 10 output steps.

DPOT [HSL+24]. In [HSL+24], DPOT is trained on a different training dataset and different evaluation metrics are used. For a fair comparison, we retrained DPOT-M and DPOT-L on our training dataset keeping the same model configurations and optimizer hyperparameters. We evaluate DPOT-M and DPOT-L using the same testing setup and metric, where rollout is used to obtain all 10 output steps.

4.B.4 Necessity of Retraining for Fairness

Different Context Window. MPP is trained using a context window of 16, while DPOT is trained using a context window of 10. As these models learn a fixed window update rule, it is neither possible nor fair to directly use MPP with a context window of 10. This also shows one benefit of BCAT's next frame prediction, i.e., flexible context window. Even though BCAT is trained with 10 input timestamps, it can directly handle longer input sequence lengths such as 16, while DPOT has to discard the first 6 timestamps.

Different Dataset. MPP and DPOT are trained on different datasets. MPP was not trained on PDEArena and CFDBench dataset, and DPOT was not trained on PDEBench incompressible NS dataset. Additionally, DPOT did not split a validation dataset, making its training dataset larger than MPP.

Different Evaluation Metric. While both MPP and DPOT report their errors using the term relative L^2 errors, the quantitative definition may differ, e.g., number of output steps, averaging in space versus space-time, etc. Thus we do not directly compare to the reported numbers in MPP and DPOT.

To address this, we retrain MPP and DPOT, using the same context window of 10, the same collection of training dataset, and evaluate them using the same metric. We utilized their code, model configuration, and hyperparameters, and only altered the necessary parts mentioned above. In Table 4.10, we compare the performance of DPOT-M and DPOT-L when we directly evaluate the pretrained checkpoints versus our retrained model. We observe that our retrained version has similar or better performance on all datasets compared to their released pretrained checkpoints.

Table 4.10: **Evaluating Pretrained Checkpoints vs. Retrained Model.** * means directly evaluating released pretrained checkpoints. Numbers in () mean that the model is not pretrained on this dataset. The numbers reported are relative L^2 errors (%). The averages are taken with respect to the 6 distinct families listed in the columns of the table.

Model	Param	PDEBench			PDEArena		CFDBench	Average
		SWE	CNS*	INS	NS	NS-cond	-	
DPOT-M	122M	0.54	1.01	5.20	4.92	8.55	0.64	3.47
DPOT-M*	122M	0.32	2.82	(81.57)	4.49	8.40	3.20	-
DPOT-L	523M	0.15	0.89	4.08	2.21	5.29	0.34	2.16
DPOT-L*	523M	0.28	2.67	(66.64)	4.34	7.23	3.12	-

4.B.5 Remarks on Single-Task Models

In Table 4.1, we compare the zero-shot performance of training a single model for all families of datasets simultaneously. The main focus is to show that these single-task models are not capable of handling more complicated tasks and datasets we are considering using a single pretrained model. For completeness, we also train these single-task models separately on each dataset, and the results are shown in Table 4.11. We observe that, the performance does improve for almost all cases (as expected); however, the error's overall order does not change. All single-task models have difficulty predicting the PDEArena dataset, suggesting larger models and better learning algorithms are needed.

4.C Visualizations

In this section, we provide visualizations of the model outputs. Figure 4.5 and Figure 4.6 contain output visualizations from the BCAT model. In Figure 4.7 and Figure 4.8, we compare different model outputs on the same example. While patch effects and patch

Table 4.11: **Single-tasked models trained on all data simultaneously v.s. on each class separately.** * means the results are obtained by training separate models on each dataset. The numbers reported are relative L^2 errors (%).

Model	Param	PDEBench			PDEArena		CFDBench
		SWE	CNS*	INS	NS	NS-cond	-
DeepONet	3.5M	3.55	7.41	64.61	35.33	51.85	12.50
DeepONet*	6× 3.5M	2.86	7.31	57.61	31.00	48.34	14.08
FNO	0.6M	3.71	6.31	36.84	38.67	55.63	8.52
FNO*	6× 0.6M	1.14	5.58	21.67	26.39	44.89	3.21
UNet	5.6M	0.33	3.19	3.43	12.56	16.82	0.76
UNet*	6× 5.6M	0.19	3.32	3.15	12.96	14.02	0.43

boundary discontinuities do exist, e.g. Figure 4.6a and Figure 4.8, which is common for patch-based methods, they are not the main source of error, e.g. high frequency and high contrast regions. To validate this, we compute the average error of interior points (masking out patch boundaries), which is 1.17% versus the total average error of 1.18%.

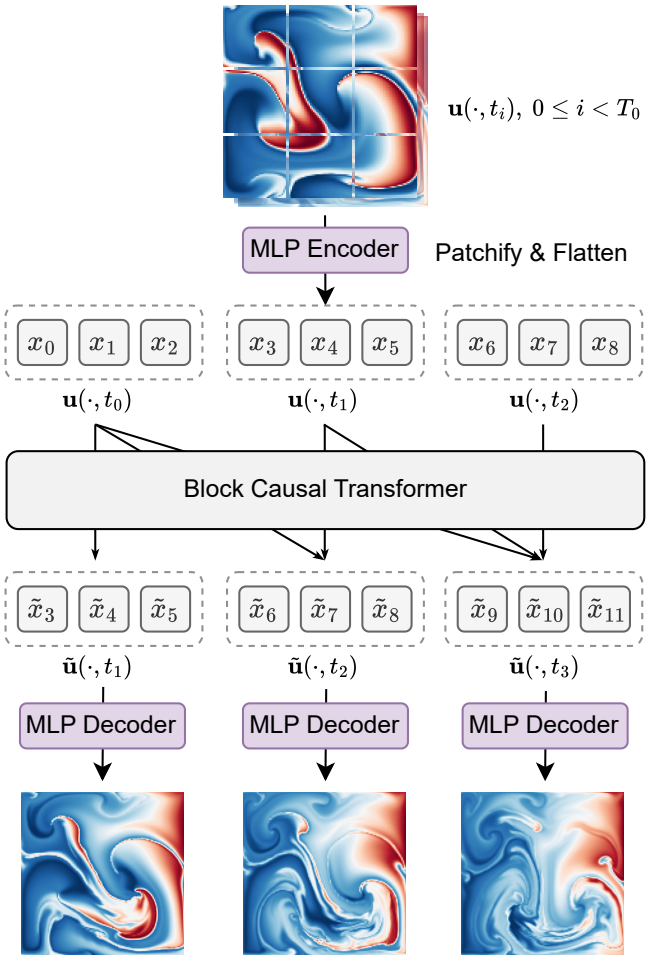


Figure 4.1: **BCAT model overview.** The inputs to the model are the initial frames sampled from the datasets, which are patchified and converted into a sequence of features. Transformer layers then take the input sequence to perform next frame prediction, where a block causal mask allows spatial interactions within a frame and temporal causality across varying-length context windows. The processed features are then transformed back to form the final predictions.

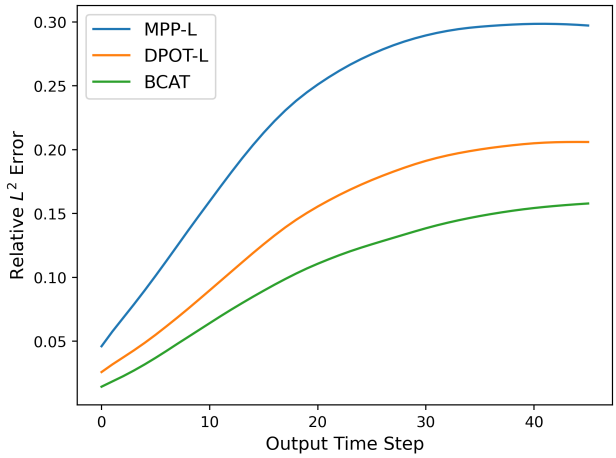


Figure 4.2: **Evaluating BCAT, DPOT-L, and MPP-L for more output time steps on PDEArena NS-cond dataset.** Rollout is used to obtain outputs beyond the training timesteps.

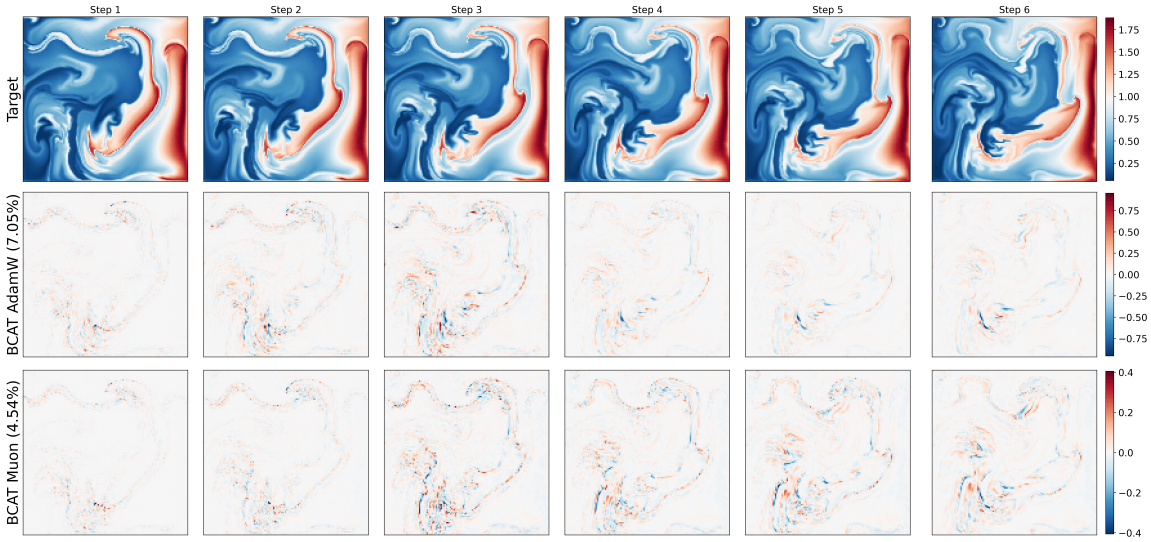


Figure 4.3: **Comparing BCAT trained with Muon vs. AdamW optimizer.** The target (first row) is the first 6 output steps from the PDEArena Navier-Stokes (conditioned) dataset (particle density channel). For each optimizer (each row), we display the difference between the target and model output. Relative L^2 errors for the full trajectories are listed after the optimizer names.

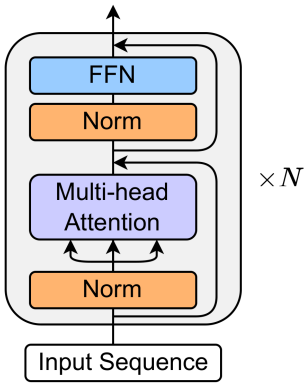


Figure 4.4: Transformer Layers used in BCAT model.

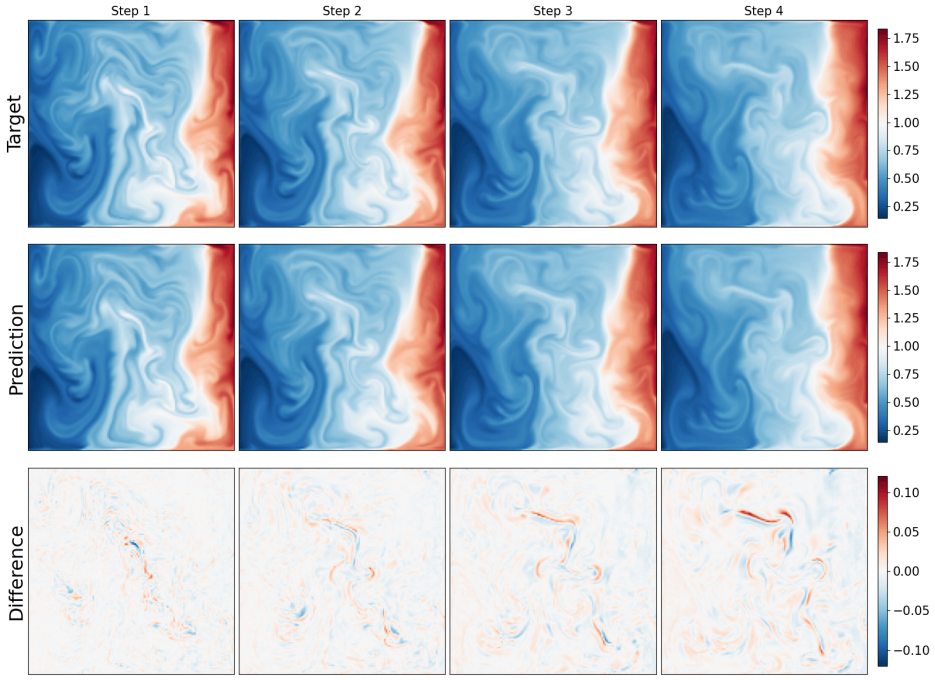
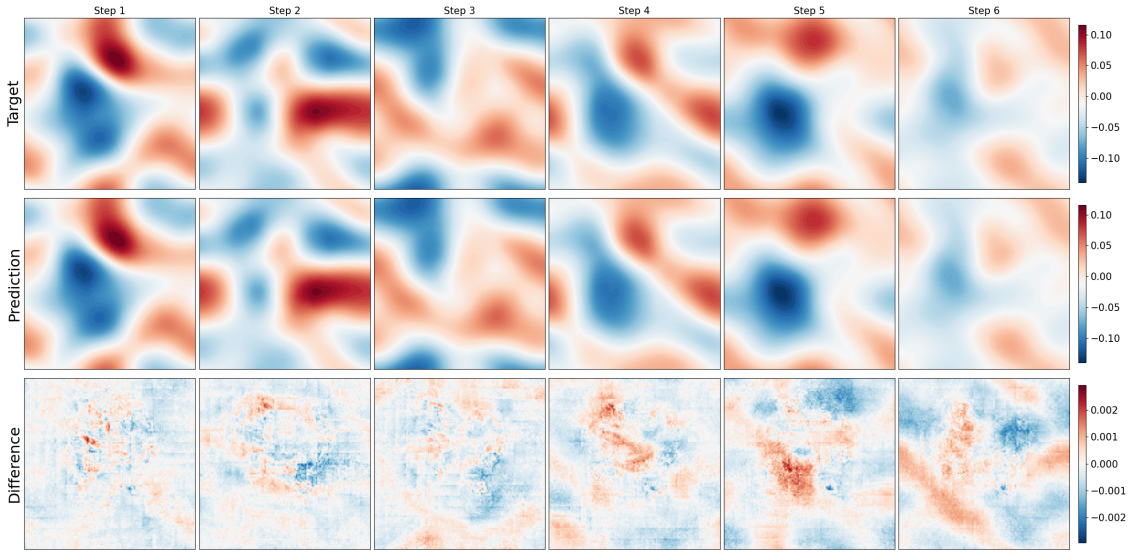
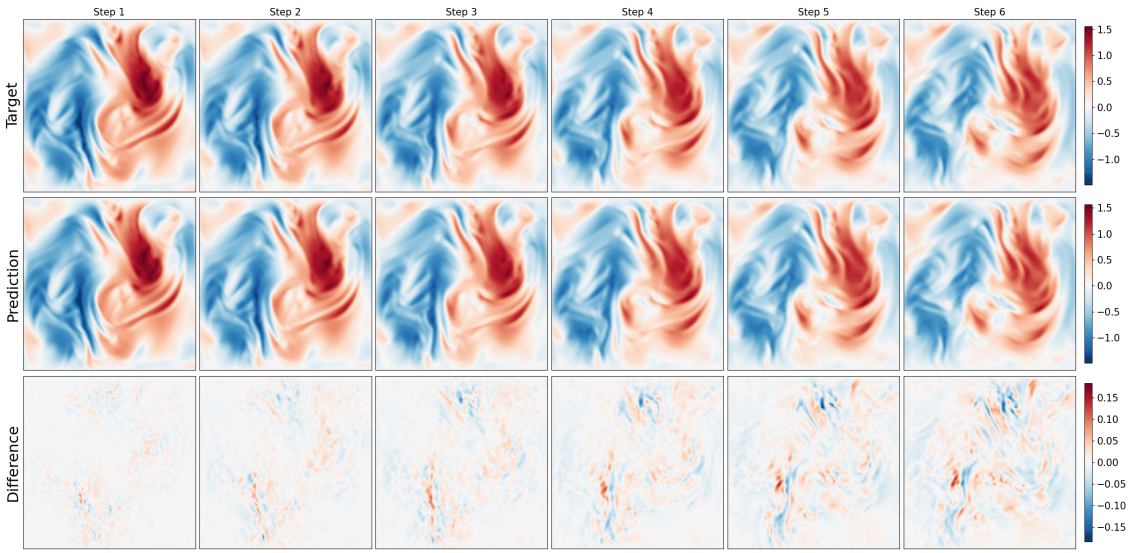


Figure 4.5: **Example outputs from the BCAT model.** 4 output steps for PDEArena Navier-Stokes dataset. The channel plotted is the particle density in equation (4.44). Each column represents a different timestamp. For this trajectory, the relative L^2 error is 1.93%.



(a) 6 output steps for PDEBench Compressible Navier-Stokes dataset. The channel plotted is the x-velocity in equation (4.41). Each column represents a different timestamp. For this trajectory, the relative L^2 error is 0.03%.



(b) 6 output steps for PDEArena Navier-Stokes (conditioned) dataset. The channel plotted is the x-velocity in equation (4.44). Each column represents a different timestamp. For this trajectory, the relative L^2 error is 4.08%.

Figure 4.6: **Two example outputs from the BCAT model.**

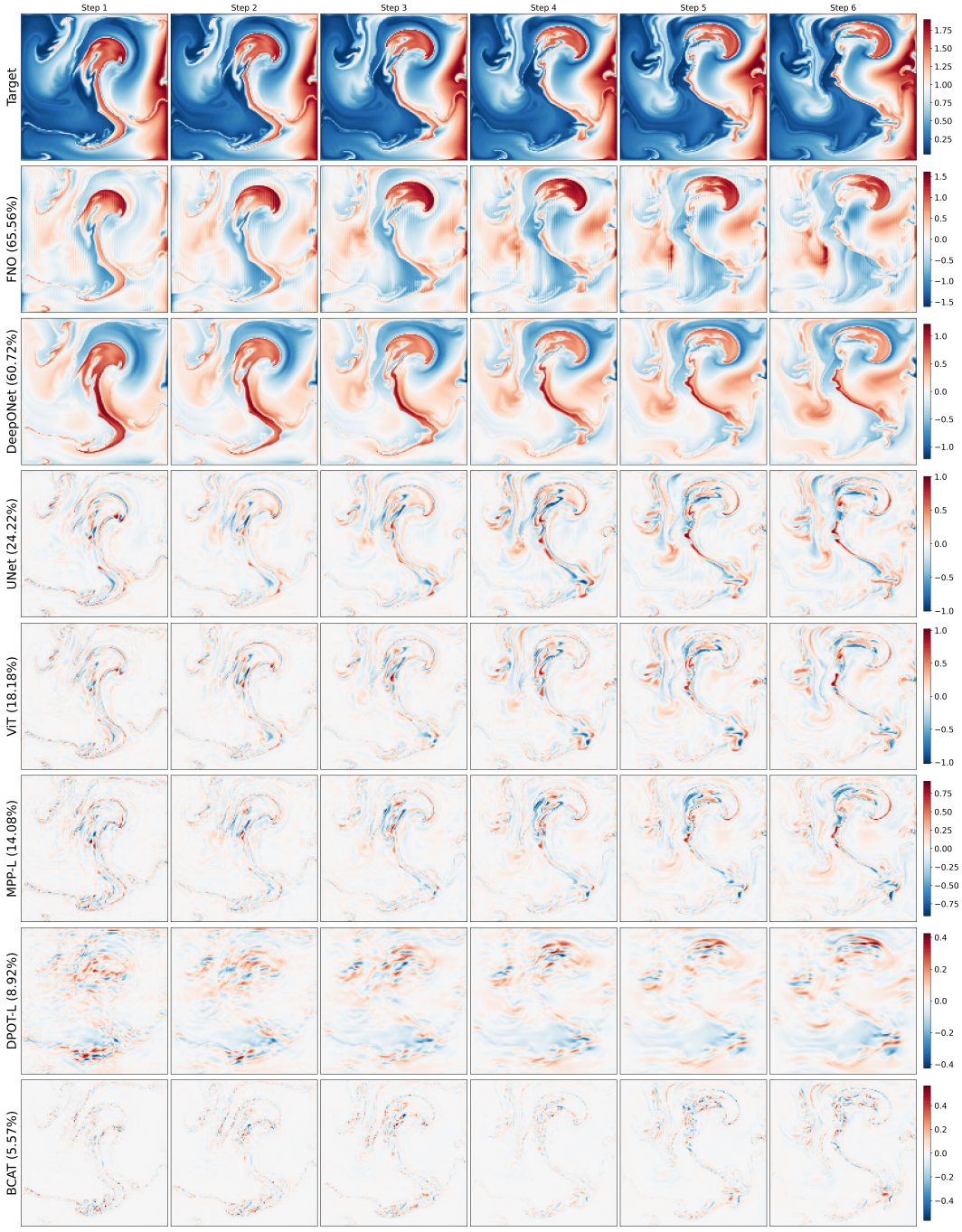


Figure 4.7: **Comparing outputs from different models.** The target (first row) is the first 6 output steps from the PDEArena Navier-Stokes (conditioned) dataset (particle density channel). For each model (each row), we display the difference between the target and model output. Relative L^2 errors for the full trajectories are listed after the model names.

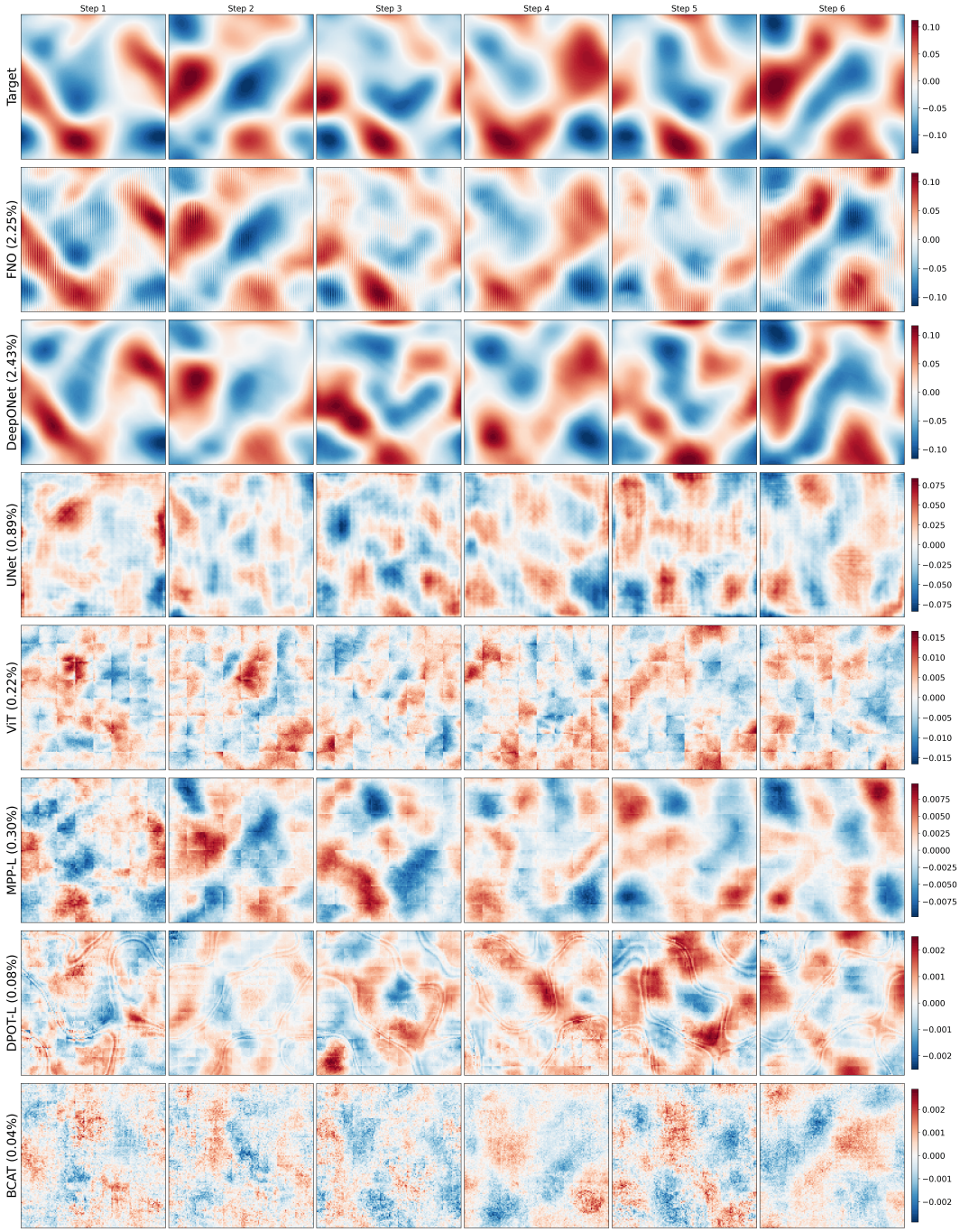


Figure 4.8: **Comparing outputs from different models.** The target is the first 6 output steps from the PDEBench Compressible Navier-Stokes dataset (y-velocity channel). For each model (each row), we display the difference between target and model output. Relative L^2 errors for the full trajectory are listed after the model names.

CHAPTER 5

Conclusion

This thesis addresses the challenges of modeling spatiotemporal dynamics with machine learning by proposing three novel, scalable PDE foundation models. By developing architectures capable of generalizing across varied physical systems, geometric structures, and temporal scales, this work advances scientific machine learning toward true multi-physics and multi-operator generalization.

In Chapter 2, we introduced *PROSE-FD*, a pretrained multimodal PDE foundation model that fuses symbolic equation representations with numerical state-variable observations. This unique modality combination demonstrated that incorporating explicit physical rules allows a single transformer-based deep neural network to perform accurate zero-shot, operator-based prediction without autoregressive drift. By simultaneously accommodating both incompressible and compressible Navier-Stokes flows, shallow water equations, and complex underlying parameter configurations, PROSE-FD establishes a versatile pathway for general-purpose surrogate modeling in two-dimensional fluid systems.

In Chapter 3, we presented *Vision In-Context Operator Networks* (VICON), an approach focused on few-shot adaptation directly from trajectory pairs, circumventing the need for explicit symbolic PDE knowledge or targeted fine-tuning. By integrating vision transformer patch-wise operations, VICON efficiently handles dense 2D fields, overcoming the computational bottlenecks for previous in-context operator learners in higher dimensions. Our comprehensive evaluations demonstrated not only superior predictive accuracy against prior multi-physics baselines but also significant robustness when faced with the irregular sampling,

missing frames, and varied timestep strides frequently seen in real-world measurements.

In Chapter 4, we proposed the *Block Causal Attention Transformer* (BCAT) for autoregressive fluid dynamics prediction. By transitioning from standard sub-frame prediction to next-frame prediction combined with a specialized block-causal mask, BCAT drastically improves computational efficiency, training parallelism, and sequential memory. We also provided theoretical guarantees under the Mori-Zwanzig formalism and universal approximation theorems, mathematically proving that the BCAT structure acts as a universal approximator for spatiotemporal dynamics. Empirically, BCAT provided state-of-the-art parameter efficiency, showing outstanding predictive capabilities on zero-shot benchmarks and strong transferability via fine-tuning to chaotic phenomena.

Together, these frameworks — multimodal operator learning, in-context prompting, and autoregressive block-causal generation — demonstrate varying routes to universally generalize machine learning models for PDEs. PROSE-FD leverages explicit equation-priors, VICON extracts implicit physical rules from immediate historical example pairs, and BCAT enforces continuous temporal causality. However, several avenues for future research remain in the pursuit of overarching foundation models for science. Extending models like VICON and BCAT effectively beyond two dimensions into dense 3D scenarios necessitates novel token-compression mechanisms, such as latent autoencoding or inter-channel attention schemas, to combat cubic memory scaling. Further investigations can explore handling irregular mesh structures or point clouds via geometric modifications, expanding applications beyond fluid mechanics into solid material prediction and molecular dynamics. Finally, addressing autoregressive error accumulation in extremely long-horizon rollouts, potentially through advanced denoising or push-forward training, will be essential to translating these surrogate models into operational real-time control setups. Ultimately, the models and principles developed in this thesis lay a versatile foundation for advancing generalized neural simulation platforms across all domains of scientific inference.

REFERENCES

[AAA+23] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[AAA+24] Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.

[AAB+24] Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J Hewett, Mojan Javaheripi, Piero Kauffmann, et al. Phi-4 technical report. *arXiv preprint arXiv:2412.08905*, 2024.

[BBL+24] Cristian Bodnar, Wessel P Bruinsma, Ana Lucic, Megan Stanley, Johannes Brandstetter, Patrick Garvan, Maik Riechert, Jonathan Weyn, Haiyu Dong, Anna Vaughan, et al. Aurora: A foundation model of the atmosphere. *arXiv preprint arXiv:2405.13063*, 2024.

[BCB14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[BGM+24] Y. Bai, X. Geng, K. Mangalam, A. Bar, A. L. Yuille, T. Darrell, J. Malik, and A. A. Efros. Sequential modeling enables scalable learning for large vision models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 22861–22872, 2024.

[BHA+21] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

[BMR+20] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[BPH+24] Tim Brooks, Bill Peebles, Connor Holmes, Will DePue, Yufei Guo, Li Jing, David Schnurr, Joe Taylor, Troy Luhman, Eric Luhman, et al. Video generation models as world simulators, 2024.

[BWT21] Gedas Bertasius, Heng Wang, and Lorenzo Torresani. Is space-time attention all you need for video understanding? *ICML*, 2(3):4, 2021.

[BWW21] J. Brandstetter, D. E. Worrall, and M. Welling. Message passing neural pde solvers. In *International Conference on Learning Representations*, 2021.

[Cao21] S. Cao. Choose a transformer: Fourier or galerkin. *Advances in neural information processing systems*, 34:24924–24940, 2021.

[CC95a] T. Chen and H. Chen. Approximation capability to functions of several variables, nonlinear functionals, and operators by radial basis function neural networks. *IEEE Transactions on Neural Networks*, 6(4):904–910, 1995.

[CC95b] Tianping Chen and Hong Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks*, 6(4):911–917, 1995.

[CCLJ23] Y. Cao, M. Chai, M. Li, and C. Jiang. Efficient learning of mesh-based physical simulation with bi-stride multi-scale graph neural network. In *International Conference on Machine Learning*, pages 3541–3558, 2023.

[CLY+26] Yadi Cao, Yuxuan Liu, Liu Yang, Rose Yu, Hayden Schaeffer, and Stanley Osher. VICON: Vision In-Context Operator Networks for Multi-Physics Fluid Dynamics Prediction. *Transactions on Machine Learning Research*, 2026.

[CRC+20] Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. Generative pretraining from pixels. In *International conference on machine learning*, pages 1691–1703. PMLR, 2020.

[CSR+24] W. Chen, J. Song, P. Ren, S. Subramanian, D. Morozov, and M. W. Mahoney. Data-efficient operator learning via unsupervised pretraining and in-context learning. *arXiv preprint arXiv:2402.15734*, 2024.

[CWPC24] Ching Chang, Wei-Yao Wang, Wen-Chih Peng, and Tien-Fu Chen. Llm4ts: Aligning pre-trained llms as data-efficient time-series forecasters. *arXiv preprint arXiv:2308.08469*, 2024.

[DBK+21] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[DFG+24] Juechu Dong, Boyuan Feng, Driss Guessous, Yanbo Liang, and Horace He. Flex attention: A programming model for generating optimized attention kernels. *arXiv preprint arXiv:2412.05496*, 2024.

[DHC+22] Y. Dang, Z. Hu, M. Cranmer, M. Eickenberg, and S. Ho. Tnt: Vision transformer for turbulence simulations. *arXiv preprint arXiv:2207.04616*, 2022.

[DJP+24] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[dKLC22] Stéphane d'Ascoli, Pierre-Alexandre Kamienny, Guillaume Lample, and Francois Charton. Deep symbolic regression for recurrent sequences. *arXiv preprint arXiv:2201.04600*, 2022.

[EKBT21] Amal Ennajar, Nadia Khouja, Rémi Boutteau, and Fethi Tlili. Deep multimodal object detection for autonomous driving. In *2021 18th International Multi-Conference on Systems, Signals & Devices (SSD)*, pages 7–11. IEEE, 2021.

[ELLZ22] Yalchin Efendiev, Wing Tat Leung, Guang Lin, and Zecheng Zhang. Efficient hybrid explicit-implicit learning for multiscale problems. *Journal of Computational Physics*, 467:111326, 2022.

[GB22] Jayesh K Gupta and Johannes Brandstetter. Towards multi-spatiotemporal-scale generalized pde modeling. *arXiv preprint arXiv:2209.15616*, 2022.

[GFQW24] Nate Gruver, Marc Finzi, Shikai Qiu, and Andrew G Wilson. Large language models are zero-shot time series forecasters. *Advances in Neural Information Processing Systems*, 36, 2024.

[GML+21] John Guibas, Morteza Mardani, Zongyi Li, Andrew Tao, Anima Anandkumar, and Bryan Catanzaro. Adaptive fourier neural operators: Efficient token mixers for transformers. *arXiv preprint arXiv:2111.13587*, 2021.

[HDPC20] Alex Henry, Prudhvi Raj Dachapally, Shubham Pawar, and Yuxuan Chen. Query-key normalization for transformers. *arXiv preprint arXiv:2010.04245*, 2020.

[HKWS19] Jonathan Ho, Nal Kalchbrenner, Dirk Weissenborn, and Tim Salimans. Axial attention in multidimensional transformers. *arXiv preprint arXiv:1912.12180*, 2019.

[HLKT25] B. Holzschuh, Q. Liu, G. Kohl, and N. Thuerey. Pde-transformer: Efficient and versatile transformers for physics simulations. *arXiv preprint arXiv:2505.24717*, 2025.

[HRR+24] Maximilian Herde, Bogdan Raonić, Tobias Rohner, Roger Käppeli, Roberto Molinaro, Emmanuel de Bézenac, and Siddhartha Mishra. Poseidon: Efficient foundation models for pdes. *arXiv preprint arXiv:2405.19101*, 2024.

[HSL+24] Zhongkai Hao, Chang Su, Songming Liu, Julius Berner, Chengyang Ying, Hang Su, Anima Anandkumar, Jian Song, and Jun Zhu. Dpot: Auto-regressive denoising operator transformer for large-scale pde pre-training. *arXiv preprint arXiv:2403.03542*, 2024.

[HSW+22] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

[HTH+24] Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, et al. Minicpm: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*, 2024.

[JJB+24] Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024.

[JLJ+23] J. Jiang, G. Li, Y. Jiang, L. Zhang, and X. Deng. Transcfd: A transformer-based decoder for flow field prediction. *Engineering Applications of Artificial Intelligence*, 123:106340, 2023.

[JSZS24] Derek Jollie, Jingmin Sun, Zecheng Zhang, and Hayden Schaeffer. Time-series forecasting, knowledge distillation, and refinement within a multimodal pde foundation model. *arXiv preprint arXiv:2409.11609*, 2024.

[JWM+23] Ming Jin, Shiyu Wang, Lintao Ma, Zhixuan Chu, James Y Zhang, Xiaoming Shi, Pin-Yu Chen, Yuxuan Liang, Yuan-Fang Li, Shirui Pan, et al. Time-llm: Time series forecasting by reprogramming large language models. *arXiv preprint arXiv:2310.01728*, 2023.

[JWY23] Zhongyi Jiang, Chunmei Wang, and Haizhao Yang. Finite expression methods for discovering physical laws from data. *arXiv preprint arXiv:2305.08342*, 2023.

[KdLC22] Pierre-Alexandre Kamienny, Stéphane d'Ascoli, Guillaume Lample, and Francois Charton. End-to-end symbolic regression with transformers. *Advances in Neural Information Processing Systems*, 35:10269–10281, 2022.

[Kin14] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KL20] Patrick Kidger and Terry Lyons. Universal approximation with deep narrow networks. In *Conference on learning theory*, pages 2306–2327. PMLR, 2020.

[KLL+23] 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*, 24(89):1–97, 2023.

[KSH+23] Thorsten Kurth, Shashank Subramanian, Peter Harrington, Jaideep Pathak, Morteza Mardani, David Hall, Andrea Miele, Karthik Kashinath, and Anima Anandkumar. Fourcastnet: Accelerating global high-resolution weather forecasting using adaptive fourier neural operators. In *Proceedings of the platform for advanced scientific computing conference*, pages 1–11, 2023.

[LBPL19] Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. *Advances in neural information processing systems*, 32, 2019.

[LCZ23] Yining Luo, Yingfa Chen, and Zhen Zhang. Cfdbench: A comprehensive benchmark for machine learning methods in fluid dynamics. *arXiv preprint arXiv:2310.05963*, 2023.

[LFBC22] M. Lino, S. Fotiadis, A. A. Bharath, and C. D. Cantwell. Multi-scale rotation-equivariant graph neural networks for unsteady eulerian fluid dynamics. *Physics of Fluids*, 34(8):087110, 2022.

[LH17] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *International Conference on Learning Representations*, 2017.

[LHLA23] Zongyi Li, Daniel Zhengyu Huang, Burigede Liu, and Anima Anandkumar. Fourier neural operator with learned deformations for pdes on general geometries. *Journal of Machine Learning Research*, 24(388):1–26, 2023.

[LJK19] Lu Lu, Pengzhan Jin, and George Em Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

[LJP+21] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

[LKA+21] Z. Li, N. B. Kovachki, K. Azizzadenesheli, B. liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021.

[LKC+24] Z. Li, N. Kovachki, C. Choy, B. Li, J. Kossaifi, S. Otta, M. A. Nabian, M. Stadler, C. Hundt, K. Azizzadenesheli, and others. Geometry-informed neural operator for large-scale 3d pdes. *Advances in Neural Information Processing Systems*, 36, 2024.

[LLC+21] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021.

[LLF24] C. Lorsung, Z. Li, and A. B. Farimani. Physics informed token transformer for solving partial differential equations. *Machine Learning: Science and Technology*, 5(1):015032, 2024.

[LMC+22] Lu Lu, Xuhui Meng, Shengze Cai, Zhiping Mao, Somdatta Goswami, Zhongqiang Zhang, and George Em Karniadakis. A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data. *Computer Methods in Applied Mechanics and Engineering*, 393:114778, 2022.

[LMF22] Zijie Li, Kazem Meidani, and Amir Barati Farimani. Transformer for partial differential equations' operator learning. *arXiv preprint arXiv:2205.13671*, 2022.

[LMZ23] Guang Lin, Christian Moya, and Zecheng Zhang. B-deeponet: An enhanced bayesian deeponet for solving noisy parametric pdes using accelerated replica exchange sgld. *Journal of Computational Physics*, 473:111713, 2023.

[LRM+21] Minghao Liu, Shengqi Ren, Siyuan Ma, Jiahui Jiao, Yizhou Chen, Zhiguang Wang, and Wei Song. Gated transformer networks for multivariate time series classification. *arXiv preprint arXiv:2103.14438*, 2021.

[LSBF23] Z. Li, D. Shu, and A. Barati Farimani. Scalable transformer for pde surrogate modeling. *Advances in Neural Information Processing Systems*, 36:28010–28039, 2023.

[LSH+24] Yuxuan Liu, Jingmin Sun, Xinjie He, Griffin Pinney, Zecheng Zhang, and Hayden Schaeffer. PROSE-FD: A Multimodal PDE Foundation Model for Learning Multiple Operators for Forecasting Fluid Dynamics. *NeurIPS Foundation Models for Science Workshop*, 2024.

[LSS25] Yuxuan Liu, Jingmin Sun, and Hayden Schaeffer. BCAT: A Block Causal Transformer for PDE Foundation Models for Fluid Dynamics. *arXiv preprint arXiv:2501.18972*, 2025.

[LSY+25] Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, et al. Muon is scalable for llm training. *arXiv preprint arXiv:2502.16982*, 2025.

[LYRK21] Ruilong Li, Shan Yang, David A Ross, and Angjoo Kanazawa. Ai choreographer: Music conditioned 3d dance generation with aist++. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13401–13412, 2021.

[LYW+24] Lei Liu, Shuo Yu, Runze Wang, Zhenxun Ma, and Yanming Shen. How can large language models understand spatial-temporal data? *arXiv preprint arXiv:2401.14192*, 2024.

[LZK+24] Z. Li, H. Zheng, N. Kovachki, D. Jin, H. Chen, B. Liu, K. Azizzadenesheli, and A. Anandkumar. Physics-informed neural operator for learning partial differential equations. *ACM/JMS Journal of Data Science*, 1(3):1–27, 2024.

[LZL+24] X. Liu, M. Zhu, L. Lu, H. Sun, and J. Wang. Multi-resolution partial differential equations preserved learning framework for spatiotemporal dynamics. *Communications Physics*, 7(1):31, 2024.

[LZM22] Paul Pu Liang, Amir Zadeh, and Louis-Philippe Morency. Foundations and trends in multimodal machine learning: Principles, challenges, and open questions. *arXiv preprint arXiv:2209.03430*, 2022.

[LZS24] Yuxuan Liu Liu, Zecheng Zhang, and Hayden Schaeffer. PROSE: Predicting Multiple Operators and Symbolic Expressions using Multimodal Transformers. *Neural Networks*, 2024.

[LZV+20] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania, and . others. Pytorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704*, 2020.

[Mor65] Hazime Mori. Transport, collective motion, and brownian motion. *Progress of theoretical physics*, 33(3):423–455, 1965.

[MRSBP+23] M. McCabe, B. Régaldo-Saint Blancard, L. H. Parker, R. Ohana, M. Cranmer, A. Bietti, M. Eickenberg, S. Golkar, G. Krawezik, F. Lanusse, and . others. Multiple physics pretraining for physical surrogate models. In *NeurIPS 2023 AI for Science Workshop*, 2023.

[NHK17] Hyeonseob Nam, Jung-Woo Ha, and Jeonghee Kim. Dual attention networks for multimodal reasoning and matching. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 299–307, 2017.

[OKS+24] O. Ovadia, A. Kahana, P. Stinis, E. Turkel, D. Givoli, and G. E. Karniadakis. Vito: Vision transformer-operator. *Computer Methods in Applied Mechanics and Engineering*, 428:117109, 2024.

[PGM+19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, and . others. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[RBL+22] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

[RDN+22] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 1(2):3, 2022.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas 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*, pages 234–241. Springer, 2015.

[RNS+18] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. *OpenAI blog*, 2018.

[RPG+21] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International conference on machine learning*, pages 8821–8831. Pmlr, 2021.

[RWC+19] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[Sch33] Günther Schulz. Iterative berechung der reziproken matrix. *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik*, 13(1):57–59, 1933.

[Sch17] Hayden Schaeffer. Learning partial differential equations via data discovery and sparse optimization. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 473(2197):20160446, 2017.

[SGGP+20] 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*, pages 8459–8468. PMLR, 2020.

[Sha20] Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

[SHK+24] Shashank Subramanian, Peter Harrington, Kurt Keutzer, Wahid Bhimji, Dmitriy Morozov, Michael W Mahoney, and Amir Gholami. Towards foundation models for scientific machine learning: Characterizing scaling and transfer behavior. *Advances in Neural Information Processing Systems*, 36, 2024.

[SLZS25] Jingmin Sun, Yuxuan Liu, Zecheng Zhang, and Hayden Schaeffer. Towards a Foundation Model for Partial Differential Equations: Multi-Operator Learning and Extrapolation. *Physical Review E*, 2025.

[SM17] Hayden Schaeffer and Scott G McCalla. Sparse model selection via integral terms. *Physical Review E*, 96(2):023302, 2017.

[SMV+19] Chen Sun, Austin Myers, Carl Vondrick, Kevin Murphy, and Cordelia Schmid. Videobert: A joint model for video and language representation learning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 7464–7473, 2019.

[SZS20] Yifan Sun, Linan Zhang, and Hayden Schaeffer. Neupde: Neural network based ordinary and partial differential equations for modeling time-dependent data. In *Mathematical and Scientific Machine Learning*, pages 352–372. PMLR, 2020.

[SZS24] Jingmin Sun, Zecheng Zhang, and Hayden Schaeffer. Lemon: Learning to learn multi-operator networks. *arXiv preprint arXiv:2408.16168*, 2024.

[TB19] Hao Tan and Mohit Bansal. Lxmert: Learning cross-modality encoder representations from transformers. *arXiv preprint arXiv:1908.07490*, 2019.

[TJY+24] Keyu Tian, Yi Jiang, Zehuan Yuan, Bingyue Peng, and Liwei Wang. Visual autoregressive modeling: Scalable image generation via next-scale prediction. *arXiv preprint arXiv:2404.02905*, 2024.

[TLI+23] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[TMG+24] Mingtian Tan, Mike A Merrill, Vinayak Gupta, Tim Althoff, and Thomas Hartvigsen. Are language models actually useful for time series forecasting? In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[TMS+23] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[TPL+22] Makoto Takamoto, Timothy Praditia, Raphael Leiteritz, Daniel MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. Pdebench: An extensive benchmark for scientific machine learning. *Advances in Neural Information Processing Systems*, 35:1596–1611, 2022.

[TRP+24] Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[WCH+24] H. Wang, Y. Cao, Z. Huang, Y. Liu, P. Hu, X. Luo, Z. Song, W. Zhao, J. Liu, J. Sun, and . others. Recent advances on machine learning for computational fluid dynamics: A survey. *arXiv preprint arXiv:2408.12171*, 2024.

[WLA+22] Gege Wen, Zongyi Li, Kamyar Azizzadenesheli, Anima Anandkumar, and Sally M Benson. U-fno an enhanced fourier neural operator-based deep-learning model for multiphase flow. *Advances in Water Resources*, 163:104180, 2022.

[WLW+24] Haixu Wu, Huakun Luo, Haowen Wang, Jianmin Wang, and Mingsheng Long. Transolver: A fast transformer solver for pdes on general geometries. *arXiv preprint arXiv:2402.02366*, 2024.

[WSS+25] S. Wang, J. H. Seidman, S. Sankaran, H. Wang, G. J. Pappas, and P. Perdikaris. CVit: Continuous vision transformer for operator learning. In *The Thirteenth International Conference on Learning Representations*, 2025.

[WWP21] S. Wang, H. Wang, and P. Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed DeepONets. *Science Advances*, 7(40):eabi8605, 2021.

[WYW+25] Feng Wang, Yaodong Yu, Guoyizhe Wei, Wei Shao, Yuyin Zhou, Alan Yuille, and Cihang Xie. Scaling laws in patchification: An image is worth 50,176 tokens and more. *arXiv preprint arXiv:2502.03738*, 2025.

[XBK+15a] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2048–2057, Lille, France, 07–09 Jul 2015. PMLR.

[XBK+15b] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057. PMLR, 2015.

[XZC23] Peng Xu, Xiatian Zhu, and David A Clifton. Multimodal learning with transformers: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[YHC+24] Z. Ye, X. Huang, L. Chen, H. Liu, Z. Wang, and B. Dong. PDEformer: Towards a foundation model for one-dimensional partial differential equations. In *ICLR 2024 Workshop on AI4DifferentialEquations In Science*, 2024.

[YLMO23] Liu Yang, Siting Liu, Tingwei Meng, and Stanley J Osher. In-context operator learning with data prompts for differential equation problems. *Proceedings of the National Academy of Sciences*, 120(39):e2310142120, 2023.

[YMLO23] Liu Yang, Tingwei Meng, Siting Liu, and Stanley J Osher. Prompting in-context operator learning with sensor data, equations, and natural language. *arXiv preprint arXiv:2308.05061*, 2023.

[YO24] L. Yang and S. J. Osher. PDE generalization of in-context operator networks: A study on 1D scalar nonlinear conservation laws. *Journal of Computational Physics*, 519:113379, 2024.

[YYZ+24] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

[ZLS23] Zecheng Zhang, Wing Tat Leung, and Hayden Schaeffer. Belnet: basis enhanced learning, a mesh-free neural operator. *Proceedings of the Royal Society A*, 479(2276):20230043, 2023.

[ZML+24a] Z. Zhang, C. Moya, W. T. Leung, G. Lin, and H. Schaeffer. Bayesian deep operator learning for homogenized to fine-scale maps for multiscale pde. *Multiscale Modeling & Simulation*, 22(3):956–972, 2024.

[ZML+24b] Z. Zhang, C. Moya, L. Lu, G. Lin, and H. Schaeffer. D2no: Efficient handling of heterogeneous input function spaces with distributed deep neural operators. *Computer Methods in Applied Mechanics and Engineering*, 428:117084, 2024.

[ZML+24c] Zecheng Zhang, Christian Moya, Lu Lu, Guang Lin, and Hayden Schaeffer. Deeponet as a multi-operator extrapolation model: Distributed pretraining with physics-informed fine-tuning. *arXiv preprint arXiv:2411.07239*, 2024.

[ZMW+25] H. Zhou, Y. Ma, H. Wu, H. Wang, and M. Long. Unisolver: PDE-conditional transformers are universal neural PDE solvers. In *ICLR 2025 Workshop on Foundation Models in the Wild*, 2025.

[ZS19] Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

[Zwa61] Robert Zwanzig. Memory effects in irreversible thermodynamics. *Physical Review*, 124(4):983, 1961.

[ZY24] Zihao Zhou and Rose Yu. Can llms understand time series anomalies? *arXiv preprint arXiv:2410.05440*, 2024.