Sorbonne Université

Doctoral School **École Doctorale Informatique, Télécommunications et Electronique (ED130)**
Joint Research Unit *Institut des Systèmes Intelligents et de Robotique*

Operator learning for modeling spatiotemporal dynamics from PDEs

---

By **Louis Serrano**

Academic Field **Computer Science**

Thesis supervised by:	Patrick Gallinari	*Supervisor*
	Jean-Noël Vittaut	*Co-supervisor*

Committee members:

Referees	Amaury Habrard	Professor at Université Jean Monnet Saint-Étienne
	Alexandros Kalousis	Professor at University of Applied Sciences, Western Switzerland
Examiners	Taraneh Sayadi	Professor at Conservatoire National des Arts et Métiers Paris
	Johannes Brandstetter	Assistant Professor at Johannes Kepler University Linz
Supervisors	Patrick Gallinari	Professor at Sorbonne Université
	Jean-Noël Vittaut	Associate Professor at Sorbonne Université

Sorbonne Université

École Doctorale Informatique, Télécommunications et Electronique (ED130)
Laboratoire *Institut des Systèmes Intelligents et de Robotique*

Apprentissage d'opérateur pour la modélisation de dynamiques spatiotemporelles issues d'EDPs

---

Par **Louis Serrano**

Thèse de doctorat d'**Informatique**

Thèse encadrée par:	Patrick Gallinari	*Directeur de thèse*
	Jean-Noël Vittaut	*Co-encadrant*

Membres du jury:

Rapporteurs	Amaury Habrard	Professeur à l'Université Jean Monnet Saint-Étienne
	Alexandros Kalousis	Professeur à la Haute École Spécialisée de Suisse Occidentale
Examinateurs	Taraneh Sayadi	Professeure au Conservatoire National des Arts et Métiers Paris
	Johannes Brandstetter	Maître de conférences à Johannes Kepler University Linz
Encadrants	Patrick Gallinari	Professeur à Sorbonne Université
	Jean-Noël Vittaut	Maître de conférences à Sorbonne Université

À Caroline et ma famille

Operator learning for modeling spatiotemporal dynamics from PDEs

Abstract

Partial differential equations (PDEs) are fundamental to scientific modeling, naturally emerging from physical principles such as mass and energy conservation. Depending on modeling assumptions, PDEs take various forms and parameterizations, yet most lack closed-form analytical solutions. Consequently, numerical methods are required to approximate solutions, converging toward the true solution as the discretization resolution increases. Over the years, scientists and engineers have developed a wide range of numerical solvers, including Finite Differences, Finite Elements, Finite Volumes, and Spectral Methods. These solvers offer theoretical guarantees: they approximate the continuous dynamics of the underlying PDEs, provide well-characterized error bounds, and can be analyzed. However, their performance is constrained by numerical stability—coarse spatial or temporal discretization can lead to instability—making them computationally expensive and time-consuming. Various statistical techniques, such as reduced-order modeling and parameterization, have been proposed to mitigate these costs, but they are not designed to generalize across a broad range of simulations.

In recent years, the increasing availability of large-scale data and computational resources has triggered a paradigm shift in fields such as computer vision and natural language processing, where deep learning models trained directly on data have outperformed traditional statistical methods relying on hand-crafted features. A similar revolution is underway in scientific computing, particularly in PDE resolution. This thesis explores how neural networks trained on data can approximate numerical solvers. These neural surrogates generate accurate PDE solutions at a fraction of the computational cost of numerical simulations, making them attractive for engineering applications. However, their major limitation is poor generalization: they struggle with changes in problem setup, such as variations in discretization, domain geometry, or PDE parameters. This thesis focuses on improving the generalization capabilities of neural solvers across these challenging scenarios. The methods developed have direct implications for industrial applications, including geometric design optimization and accelerating numerical simulations. This manuscript presents the main contributions of this thesis, structured as follows:

(i) We introduce a novel framework based on neural fields for operator learning, addressing constraints on mesh discretization. Coral, our framework, enables dynamics modeling and geometry-aware inference on data discretized over regular grids, irregular grids, and even point clouds. (ii) We propose Infinity, a specialized version of Coral tailored for solving Reynolds-Averaged Navier-Stokes (RANS) equations. Unlike Coral, Infinity incorporates both geometrical features (e.g., signed distance functions) and flow parameters (e.g., inlet velocities) to predict the steady-state solution. We validate its performance on a dataset representative of real–world computational fluid dynamics challenges. (iii) We introduce Aroma, an im-

proved neural surrogate, that extends Coral and existing transformer-based methods. Aroma employs an encoder-decoder architecture that maps observations to latent tokens while incorporating information about domain geometry. To enhance stability at inference, we integrate a diffusion-based transformer for modeling temporal dynamics. (iv) Inspired by the in-context learning capabilities of large language models, we propose Zebra, a novel framework for solving PDEs. Zebra adapts trajectory forecasts to various conditioning prompts, including past observations or similar trajectories. The model employs vector quantization to discretize observations into a physics-informed vocabulary and uses a decoder-only transformer trained with a next-token objective. We design a data pipeline that pretrains Zebra on diverse dynamical tasks, enabling robust generalization.

Apprentissage d'opérateur pour la modélisation de dynamiques spatiotemporelles issues d'EDPs

Résumé

Les équations aux dérivées partielles (EDP) sont omniprésentes en modélisation scientifique. Elles émergent naturellement des premiers principes physiques comme la conservation de la masse ou de l'énergie. Selon les hypothèses, elles peuvent prendre diverses formes et paramétrisations, mais n'admettent que rarement une solution analytique. Il est donc nécessaire de recourir à des méthodes numériques (différences finies, éléments finis, volumes finis, méthodes spectrales) pour obtenir des solutions approchées. Ces méthodes offrent des garanties théoriques en termes de convergence et d'analyse d'erreur, toutefois leur stabilité dépend fortement de la finesse de la discrétisation, ce qui engendre des coûts de calcul importants. Bien que certaines méthodes statistiques, comme la réduction d'ordre ou la paramétrisation, tentent de réduire ces coûts, elles restent peu adaptables à une diversité de configurations.

L'accroissement des données disponibles et des ressources de calcul a récemment bouleversé des domaines comme la vision par ordinateur ou le traitement du langage naturel, où les modèles d'apprentissage profond surpassent les approches statistiques classiques basées sur des représentations conçues par des experts. Une tendance similaire émerge dans le domaine des simulations scientifiques, notamment pour la résolution d'EDP. Cette thèse étudie comment des réseaux de neurones entraînés sur des données issues de simulations numériques peuvent servir de modèles substituts capables de générer des solutions précises à faible coût. Ces modèles sont particulièrement prometteurs pour les applications industrielles, mais leur principal défaut est qu'ils s'adaptent mal aux changements dans la discrétisation des conditions initiales, dans la géométrie du domaine ou dans la paramétrisation de l'EDP.

Ce manuscrit explore différentes approches pour améliorer la généralisation face à ces scénarios, et expose les principales contributions de cette thèse, qui portent sur les sujets suivants :

(i) Nous introduisons une nouvelle méthodologie basée sur les champs neuronaux pour l'apprentissage d'opérateur. Cette approche, appelée Coral, allège les contraintes liées au maillage de discrétisation des données. Elle peut être utilisée pour modéliser des dynamiques spatiotemporelles ou inférer des solutions stationnaires, sur des grilles régulières, irrégulières ou des nuages de points. (ii) Nous proposons Infinity, une version spécialisée de Coral pour la résolution des équations de Reynolds-Averaged Navier-Stokes (RANS). Contrairement à Coral, Infinity est conçu pour prendre en entrée à la fois des caractéristiques géométriques (par exemple, une fonction de distance signée) et des paramètres d'écoulement (par exemple, des vitesses d'entrée) pour prédire la solution stationnaire du système. Nous avons mené des expériences sur des cas représentatifs des défis rencontrés en mécanique des fluides numérique. (iii) Nous avons ensuite développé Aroma, un modèle substitut neuronal qui étend Coral et les modèles basés sur les transformeurs. Aroma repose sur une architecture encodeur-décodeur qui projette les observations vers une séquence de

tokens latents en tenant compte de la géométrie du domaine. Pour la modélisation temporelle, nous utilisons un transformeur avec un mécanisme de débruitage pour accroître la stabilité des simulations. (iv) Nous nous sommes inspirés des capacités des modèles de langage à réaliser un apprentissage en contexte pour de nouvelles tâches, et proposons Zebra, une méthode qui permet d'adapter la prédiction des trajectoires à divers types de conditionnement, incluant des observations passées ou des trajectoires similaires. Le modèle utilise la quantification vectorielle pour discrétiser les observations en un vocabulaire physique, et un transformeur entraîné pour prédire le prochain token. Nous proposons un pipeline de données intégrant une diversité de tâches dynamiques pour améliorer la généralisation du modèle.

Remerciements

Tout d'abord, je tiens à remercier mes encadrants. Patrick, merci de m'avoir offert l'opportunité de réaliser cette thèse sous ta direction, et de m'avoir accordé la liberté d'explorer mes propres pistes de recherche. Je t'en suis profondément reconnaissant. Ton aide précieuse tout au long de ces années a été déterminante. Rédiger un article scientifique demande un travail colossal, et il est évident que tes conseils, ta vision et ton sens du détail ont grandement contribué au bon déroulement de cette thèse.

Jean-Noël, après notre déménagement à la Pyramide, nous avons eu moins d'occasions d'échanger, mais tu m'as toujours été d'un grand soutien. Je ne pense pas que j'aurais pu soumettre la première version de Coral sans ton aide. Merci également pour les cours, et pour ton soutien pour affronter les méandres administratifs de la Sorbonne.

Je tiens aussi à remercier toutes les personnes de l'ISIR qui m'ont aidé au fil de la thèse. Merci bien sûr à Christophe, pour ton aide quotidienne sur les GPUs, et pour tes excellents conseils au babyfoot. J'espère pour toi que le PSG décrochera enfin la Ligue des Champions cette année ! Merci également à Awatef, pour ton aide dans l'organisation de la soutenance.

Je remercie évidemment tous les membres du jury d'avoir accepté de participer à cette soutenance, et tout particulièrement les rapporteurs, Amaury et Alexandros, pour leurs retours précieux sur le manuscrit. Merci également à Johannes pour cette belle opportunité que tu m'as offerte pour la suite de mon parcours. J'ai hâte de commencer cette nouvelle aventure.

Je souhaite aussi remercier l'ensemble de l'équipe MLIA, ainsi que tous les doctorants que j'ai eu la chance de croiser au cours de ces trois ans et demi. Une pensée particulière pour l'équipe spatiotemp : Emmanuel, Jean-Yves, Matthieu, Ramón Daniel, Jorge, Joceran, Biel. Mais plus encore, je remercie celles et ceux avec qui j'ai eu le plaisir de collaborer de près : cette thèse n'aurait pas été la même sans vous.

Yuan, merci d'avoir partagé ton expérience et tes conseils tout au long de la thèse. Ton aide sur la rédaction des papiers a été précieuse, et nos discussions, sur la recherche ou autre, ont toujours été enrichissantes. Tu as été un véritable modèle

pour moi.

Lise et Armand, merci pour votre énergie et votre soutien tout au long de ces années. Je vous dois une fière chandelle, et je vous souhaite le meilleur pour la suite.

Merci à toi, Léon, pour notre collaboration sur Infinity et Timeflow, et pour m'avoir embarqué dans le projet avec Étienne.

Étienne, je sais que je t'ai cassé les pieds avec tous les papiers que je t'envoyais, mais merci pour tout. Tu as été une oreille attentive dans les moments difficiles, et je suis très heureux d'avoir mené ce projet avec toi. C'est l'un de mes meilleurs souvenirs de la thèse.

Lise, merci de m'avoir permis d'explorer ta direction de recherche. Je te suis sincèrement reconnaissant. J'aurais aimé t'aider davantage sur ce papier, et je regrette de ne pas avoir eu le temps de le faire.

Merci à toi, Pierre, pour m'avoir expliqué les bases des LLMs, et pour ton aide sur Zebra. D'ailleurs, c'est toi qui as trouvé ce nom : j'espère que ce petit coup de marketing portera ses fruits !

Armand, encore merci pour ton aide cruciale sur Zebra. Sans toi, cela n'aurait pas été possible.

Merci aussi à Adel et Louis, j'espère que mon expertise en profiling vous aura été utile pour ACCO !

Et enfin, un grand merci à Paul et Loïck. Vous m'avez rendu un peu fou ces derniers mois avec Jafar, mais grâce à vous, j'apprécie encore plus les features profonds.

Une pensée particulière pour mes amis Ruben, Nicolas, Sam, Thomas et William. Je suis vraiment heureux d'avoir pu rester proche de vous pendant la thèse. J'ai perdu de vue pas mal de monde au fil des années, mais avec vous, je sais qu'on pourra toujours compter les uns sur les autres, c'est ce qui fait la force de notre groupe. On se retrouve bientôt pour un Catane ou un Abyss !

Je tiens à remercier chaleureusement ma famille. Merci Papa, merci Maman, de m'avoir toujours soutenu dans mes choix, et d'avoir été là pour moi dans les moments plus compliqués. Je sais que vous n'étiez pas très rassurés quand je vous ai annoncé que je voulais faire une thèse, mais je crois que vous pouvez être sereins maintenant : le plus dur est passé ! Merci à Gabriel et Paul pour toutes ces sessions de padel et de tennis ensemble. Elles m'ont fait un bien fou. Et merci à toi, Frédéric, pour ta sagesse et ton esprit philosophe. Je sais que tu es bien occupé depuis la naissance de Camille, mais il faudra qu'on s'organise une partie de badminton dès que tu as un moment !

Et enfin, je voulais te remercier, toi, Caroline, du fond du cœur, pour tous les moments que nous avons partagés pendant cette thèse. On a vécu cette aventure

côte à côte, toi l'internat et moi la thèse, et tu as été un soutien inébranlable, et une source de bonheur incroyable pendant ces trois années. On a traversé des périodes parfois difficiles, mais on s'est toujours relevés ensemble. Un grand merci aussi à ta famille, pour m'avoir toujours accueilli avec générosité et bienveillance à Nancy.

Je repense avec émotion aux treks que nous avons faits ensemble ; je suis tellement heureux que tu aies partagé ta passion avec moi. Ces escapades, hors du temps, nous ont permis de nous retrouver tous les deux, et elles comptent parmi mes plus beaux souvenirs de ces dernières années.

Je ne te remercierai jamais assez pour ton aide précieuse tout au long de cette période. Ce n'était pas simple, et pourtant tu m'as toujours encouragé, même pendant mes longues soirées à lancer des expériences, ou mes week-ends passés à analyser des features.

Merci pour tout. Tu vas beaucoup me manquer aux États-Unis durant ces quelques mois, et j'ai déjà hâte de rentrer pour commencer une nouvelle vie avec toi. Qui sait, peut-être à Lyon ? À chaque jour suffit sa peine, comme dirait Ruben.

Symbols

The manuscript consistently employs the following symbols, with additional notations introduced as needed in specific chapters or sections.

Domain	Symbol	Description
PDE	u	PDE solution
	u^0	initial condition
	u^t	PDE solution at time t
	a	input function
	f	forcing term
	g	dynamical model
	$\mathcal{B}$	boundary condition
	$\mathcal{T}_{in}$	Training horizon
	t	Temporal coordinate
	Ω	Spatial domain
	$\mathcal{X}$	Spatial grid
	x	Spatial coordinate
	$\mathbb{A}$	Symbolic values support
	c	Number of input channels
General	i, j	Samples index
	n	Number of samples
	$\mathcal{L}$	Loss
	$\mathbf{W}$, $\mathbf{b}$	Weights and biases of a linear layer
	$p(\mathbf{x})$	Probabilistic distribution
	$p_\theta(\mathbf{x})$	Probabilistic distribution approximation
	$\mathbb{E}_x$	Expectation over x
	$\mathcal{N}(\mu; \sigma^2)$	Gaussian distribution with mean μ and standard deviation σ
	∇	Gradient
	π	Sampling rate
Neural Networks	θ, θ'	Weights of a neural network
	$h_{\mathbf{w}}$	Hypernetwork
	α	inner learning rate
	η	outer learning rate
	$\mathcal{B}$	Batch

Contents

List of Figures

List of Tables

Part I

Research Context

Chapter 1

Introduction

1.1 Context and challenges

Modeling physical, chemical or biological systems with partial differential equations (PDEs) is a cornerstone of science and engineering. Unfortunately, obtaining solutions to these equations is not trivial, and except under simplifying assumptions it is not possible to obtain them analytically. Therefore, *numerically* solving PDEs with *solvers* is the standard way to find approximate solutions (Olver, 2007). These solvers implement a discretized version of the original PDE formulation, and come with various convergence guarantees and error bounds on the approximation error. Given the highly diverse nature of physical phenomena and range of applications, there exists a variety of solvers, designed for some particular systems (Quarteroni and Valli, 2008). For example, finite differences are usually restricted to regular domains, while finite elements are a good default choice for systems with irregular boundaries and geometries. There is also a trade-off between accuracy or stability and efficiency of the method, which means that the amount of computational resource is a key criterion for the choice of numerical scheme. The stability of a solver can depend on key parameters of the equation itself (Courant et al., 1967),

but also on the conditions at the boundary and even the initial condition (Hairer and Wanner, 2010). All these factors make the choice of a solver a case-by-case decision.

Solving a PDE enables to gain knowledge over a particular system or model. It can be used, among others, to achieve temporal forecasting (e.g. in weather modeling), design exploration and optimization (e.g. in aerodynamics), and inference of parameters (e.g. in epidemic modeling) (Ohana et al., 2024). For example, solving the equations of the atmosphere (Richardson and Lynch, 1922; Simmons et al., 1989) lets us determine how the system will evolve over time and therefore performs a *forecast*. Similarly, solving the RANS equation around a cylinder or an airfoil leads to a *prediction* of the steady-state of the system and thus to an assessment of key properties of the design (Bonnet et al., 2022). Instead of building and testing real prototypes, which can be costly and inefficient, engineers can *optimize* designs with in-silico simulations (Catalani et al., 2024). This can drastically accelerate the exploration and evaluation of new designs for complex industrial systems. Finally, aligning numerical simulations with observational data allows scientists to *infer* unknown modeling parameters (Cranmer et al., 2019).

1.2 Motivations for Using Deep Learning to Solve PDEs

In this manuscript, we will present existing deep learning (Goodfellow, 2016) methods for solving PDEs as well as introduce novel frameworks to address key challenges. Before delving into the specific research objectives and context of our work, we first provide an intuition as to why deep learning holds potential in solving PDEs.

As mentioned above, numerical solvers are powerful tools for obtaining PDE solutions, yet they come with notable drawbacks. Namely, they require substantial expertise to work in practice and are computationally expensive, especially for high-dimensional problems. While hardware advancements can address memory limitations to some extent, reducing simulation times remains a persistent challenge. Therefore, designing an efficient solver requires a deep understanding of the problem's physics, as well as careful choices in numerical schemes, which is critical to balance efficiency and accuracy. Factors such as the equation's stiffness, the domain resolution, and the temporal or spatial ranges of interest are critical for this matter.

However, even with carefully tuned solvers, running simulations that resolve all scales of a problem (known as direct numerical simulations, or DNS) remains computationally expensive. DNS seeks to capture all scales and their interactions, which makes the simulations highly accurate but impractical for many applications (Ferziger and Peric, 1996).

To mitigate computational costs, researchers often turn to reduced-order models

(ROMs, Brunton and Kutz (2019)) and parametrization techniques such as Large Eddy Simulation (LES) and Reynolds-averaged Navier-Stokes (RANS). These methods simplify the problem either by projecting the system onto a lower-dimensional space, as in Proper Orthogonal Decomposition (POD), or by modeling unresolved scales using closure terms.

However, these approaches come with limitations. Classical ROMs, such as those based on POD, are typically constructed using the first snapshots of a trajectory and rely on linear dimensionality reduction techniques. As a result, they primarily capture the dominant modes present in the early stages of the system, which may limit their ability to accurately represent the dynamics at later time steps. Additionally, in the classical setting, ROMs are usually built from a single trajectory or a restricted set of conditions rather than learning representations across multiple trajectories. This lack of cross-trajectory learning makes them highly dependent on the specific conditions used during the basis construction, reducing their ability to generalize to new scenarios.

Similarly, closure terms in LES and RANS are often derived from empirical studies or theoretical approximations rather than being systematically learned from data. These models rely on heuristics that may perform well under specific conditions but fail to generalize across different flow regimes. Tuning these parameters to align simulations with experimental data can require significant manual effort and iterative calibration. For instance, in weather modeling, handcrafted parameterizations have been shown to be less efficient than those learned entirely through deep learning (Kochkov et al., 2023).

Deep learning offers the ability to learn representations directly from simulation data, making it possible to develop more adaptive and generalizable modeling approaches. Over the past decade, deep learning has transformed fields such as natural language processing and computer vision by training models on large-scale datasets, where it has consistently outperformed traditional methods that rely on handcrafted features. This success suggests that similar data-driven approaches could be leveraged to solve PDEs more efficiently.

Deep learning has already been applied to PDE modeling in various ways, which can be categorized as follows: (i) *Physics informed deep learning*: These methods replace traditional solvers entirely by formulating instance-specific optimization problems, where neural networks act as function approximators and leverage automatic differentiation to directly solve PDEs. (ii) *Hybrid modeling*: Instead of replacing numerical solvers, this approach integrates neural networks as parameterizations to improve existing models, helping to capture unresolved scales or model partially known physical processes. (iii) *Data-driven neural surrogates*: These models learn to approximate the time-stepping process of a PDE, providing a significantly faster alternative to numerical solvers. They are trained on data without requiring prior knowledge of the governing equations.

1.3 Research Objectives

This thesis focuses on data-centric approaches, specifically neural surrogates, which align closely with the deep learning paradigm that has achieved remarkable success in text and image processing. However, key challenges arise when applying these methods to PDEs. In this work, we address the two main following aspects:

1. **Improving generalization with respect to changes in domain discretization and geometry**
    - **Handling irregular domains** Neural surrogates must generalize beyond regular grids to varying domains and discretizations. Many CFD applications, such as airfoil or car design optimization, involve irregular meshes. Similarly, weather models rely on sparse and unevenly spaced spatiotemporal observations. Existing neural architectures are not well-suited for such settings.
    - **Encoding domain geometry** To generalize across geometries, a neural surrogate should integrate essential information about the domain and its boundary conditions. Current architectures typically rely on precomputed features to encode this information.
2. **Improving generalization with respect to new dynamics**
    - **Adaptive solutions at inference** A neural surrogate should be versatile across different PDEs, capable of handling variations in coefficients, forcing terms, and boundary conditions. It should adapt seamlessly to these changes at inference without requiring finetuning.
    - **Uncertainty quantification** In the presence of varying or previously unseen dynamics, reliable neural surrogates should quantify the uncertainty of their predictions. Standard surrogates typically regress the field values at the next time step, effectively learning to predict the posterior mean. However, real-world applications demand not just a mean estimate, but also a measure of confidence.

1.4 Core Contributions

In this manuscript, we take a step-by-step approach to tackling these challenges. Our contributions address key aspects of these problems, laying a solid foundation for further exploration and expansion.

(i) We present **Coral**, a novel operator learning framework based on neural fields, designed to overcome limitations related to discretization meshes. Coral enables geometry-aware dynamics modeling on data discretized over regular grids,

irregular meshes, and even unstructured point clouds. (ii) We introduce **Infinity**, a specialized variant of Coral tailored for solving Reynolds-Averaged Navier-Stokes (RANS) equations. Infinity extends Coral by incorporating geometric features (e.g., signed distance functions) and flow parameters (e.g., inlet velocities). We evaluate its performance on datasets representative of real-world CFD challenges. (iii) We develop **Aroma**, an enhanced neural surrogate that builds upon Coral and existing transformer-based approaches. Aroma employs an encoder-decoder architecture that maps observations to latent tokens while encoding domain geometry. To improve temporal stability during inference, it leverages a diffusion-based transformer to model time evolution. (iv) Drawing inspiration from the in-context learning capabilities of large language models, we propose **Zebra**, a novel framework for solving PDEs via example-based conditioning. Zebra adapts trajectory forecasts to various prompts, such as past observations or similar trajectories. It uses vector quantization to discretize observations into a physics-informed vocabulary and a decoder-only transformer trained with a next-token objective. A dedicated data pipeline enables pretraining on diverse dynamical systems, fostering strong generalization.

These contributions, which will be discussed in detail in Part II, are summarized below.

1.4.1 Meshless operators

Operator Learning with Neural Fields: Tackling PDEs on General Geometries. Machine learning approaches for solving partial differential equations require learning mappings between function spaces. While convolutional or graph neural networks are constrained to discretized functions, neural operators present a promising milestone toward mapping functions directly. Despite impressive results they still face challenges with respect to the domain geometry and typically rely on some form of discretization. In order to alleviate such limitations, we present CORAL, a new method that leverages coordinate-based networks for solving PDEs on general geometries. CORAL is designed to remove constraints on the input mesh, making it applicable to any spatial sampling and geometry. Its ability extends to diverse problem domains, including PDE solving, spatio-temporal forecasting, and geometry-aware inference. CORAL demonstrates robust performance across multiple resolutions and performs well in both convex and non-convex domains, surpassing or performing on par with state-of-the-art models.

Serrano, L., Le Boudec, L., Kassaï Koupaï, A., Wang, T. X., Yin, Y., Vittaut, J. N., Gallinari, P. (2023). Operator learning with neural fields: Tackling pdes on general geometries. Neurips 2023.

Infinity: Neural Field Modeling for Reynolds-Averaged Navier-Stokes Equations For numerical design, the development of efficient and accurate surrogate models is paramount. They allow us to approximate complex physical phenomena, thereby reducing the computational bur- den of direct numerical simulations. We propose INFINITY, a deep learning model that utilizes implicit neural representations (INRs) to address this challenge. Our framework encodes geometric information and physical fields into compact representations and learns a mapping between them to infer the physical fields. We use an airfoil design optimization problem as an example task and we evaluate our approach on the challenging AirfRANS dataset, which closely resembles real-world industrial use-cases. The experimental results demonstrate that our framework achieves state-of-the-art performance by accurately inferring physical fields throughout the volume and surface. Additionally we demonstrate its applicability in contexts such as design exploration and shape optimization: our model can correctly predict drag and lift coefficients while adhering to the equations.

Serrano, L., Migus, L., Yin, Y., Mazari, J. A., Gallinari, P. (2023). Infinity: Neural field modeling for reynolds-averaged navier-stokes equations. ICML 2023 SynS ML workshop.

AROMA: Preserving Spatial Structure for Latent PDE Modeling with Local Neural Fields. We present AROMA (Attentive Reduced Order Model with Attention), a novel framework for modeling partial differential equations that combines the strengths of transformers and local neural fields. AROMA features a flexible encoder-decoder architecture composed of cross-attention and self-attention blocks, enabling the extraction of smooth latent representations from spatial physical fields across diverse input formats, including irregular grids and point clouds. This versatility removes the need for arbitrary patching schemes typically used with transformers, facilitating efficient processing of complex geometries. The sequential structure of the latent representation can be interpreted spatially and supports the use of a conditional transformer to model temporal dynamics. By adopting a diffusion-based formulation, AROMA improves inference stability and enables longer, reliable rollouts compared to conventional MSE-based training. We demonstrate AROMA's effectiveness in simulating 1D and 2D PDEs, showcasing its ability to capture complex dynamical behaviors with high fidelity.

Serrano, L., Wang, T. X., Naour, E. L., Vittaut, J. N., Gallinari, P. (2024). AROMA: Preserving Spatial Structure for Latent PDE Modeling with Local Neural Fields. Neurips 2024.

1.4.2 In-context adaptation for solving PDEs

Zebra: In-Context and Generative Pretraining for Solving Parametric PDEs Solving time-dependent parametric partial differential equations is challenging, as models must adapt to variations in parameters such as coefficients, forcing terms, and boundary conditions. Data-driven neural solvers either train on data sampled from the PDE parameters distribution in the hope that the model generalizes to new instances or rely on gradient-based adaptation and meta-learning to implicitly encode the dynamics from observations. This often comes with increased inference complexity. Inspired by the in-context learning capabilities of large language models (LLMs), we introduce Zebra, a novel generative auto-regressive transformer designed to solve parametric PDEs without requiring gradient adaptation at inference. By leveraging in-context information during both pre-training and inference, Zebra dynamically adapts to new tasks by conditioning on input sequences that incorporate context trajectories or preceding states. This approach enables Zebra to flexibly handle arbitrarily sized context inputs and supports uncertainty quantification through the sampling of multiple solution trajectories. We evaluate Zebra across a variety of challenging PDE scenarios, demonstrating its adaptability, robustness, and superior performance compared to existing approaches.

Serrano, L., Koupaï, A. K., Wang, T. X., Erbacher, P., Gallinari, P. (2024). Zebra: In-Context and Generative Pretraining for Solving Parametric PDEs. ICML 2025.

1.5 Other contributions

Here, two contributions are mentioned that are not detailed in this manuscript, but were carried out during the course of this thesis. The first is a collaboration with Etienne Le Naour, whose PhD focused on time-series forecasting and imputation. We found that the method I developed in Coral could be effectively transferred to his domain of expertise, and we jointly explored this extension of the framework. This collaboration resulted in a publication in Transactions on Machine Learning Research (TMLR) in 2024.

Time series continuous modeling for imputation and forecasting with implicit neural representations. In this contribution, we present a novel modeling approach called "TimeFlow" for time series imputation and forecasting that addresses challenges in real-world data, such as irregular samples, missing data, and unaligned measurements from multiple sensors. Our method utilizes a continuous-time model of the series' evolution dynamics and incorporates conditional, implicit neural representations for sequential data. A modulation mechanism, driven by

meta-learning, allows adaptation to new samples and long-term extrapolation beyond observed time windows. This model offers a flexible and unified framework for both imputation and forecasting tasks in diverse scenarios. It demonstrates state-of-the-art performance on classic benchmarks and surpasses other time-continuous models.

Le Naour, E., Serrano, L., Migus, L., Yin, Y., Agoua, G., Baskiotis, N., Gallinari, P., and Guigue, V. **Time Series Continuous Modeling for Imputation and Forecasting with Implicit Neural Representations.** Transactions on Machine Learning Research (TMLR) 2024.

The second contribution is an early version of Zebra, in which I explored replacing the discrete tokenization and transformer architecture commonly used in large language models – later adopted in the final version of Zebra – with a continuous distribution-based approach. This method did not achieve the desired performance, and it was uncertain at the time whether it could support in-context learning within the proposed architecture, therefore I ultimately returned to a discrete formulation for Zebra. Nevertheless, this preliminary investigation opened a new research direction that is now being actively investigated within the team.

Zebra: A Continuous Generative Transformer for Solving Parametric PDEs Tokenization is a cornerstone of language modeling, enabling the discretization of values into a finite vocabulary. This approach has demonstrated exceptional scalability across language, image, and video generation tasks. However, for spatial modalities such as images, videos, and physical fields, discrete encoders often produce latent representations with lower reconstruction accuracy compared to their continuous counterparts. In this work, we introduce a continuous version of Zebra: an auto-regressive transformer operating in a continuous domain, with its output distribution parameterized by a mixture of Gaussians. Pre-trained on specific families of PDEs, Zebra excels in dynamic forecasting, outperforming existing neural operators and solvers. Furthermore, it paves the way for the development of foundation models extensively pre-trained on diverse PDE scenarios, enabling effective solutions to PDE challenges with limited data.

Serrano, L., Erbacher, P. , Vittaut, J.N., Gallinari, P. (2024). Zebra: a continuous generative transformer for solving parametric PDEs. ICLR 2024 AI4PDE workshop.

1.6 Structure of the thesis

This thesis is organized as follows:

- Chapter 2 provides an overview of the fundamental concepts of partial differential equations and the numerical techniques used in traditional solvers.
- Chapter 3 covers the foundations of deep learning.
- Chapter 4 sheds light on existing methods for tackling the resolution of PDEs.
- Chapter 5 presents our first contribution, which leverages neural fields to solve PDEs. We validate the proposed framework through experiments on various tasks, including initial value problems, dynamics modeling, and geometry-aware inference.
- Chapter 6 extends the previous neural field framework to address RANS equations around airfoil geometries, with a particular focus on learning representations of shape and boundary conditions.
- Chapter 7 introduces a framework utilizing attention mechanisms to handle irregular grids and geometries in complex datasets. This chapter emphasizes modeling dynamics in such scenarios.
- Chapter 8 outlines our final contribution, focused on solving parametric PDEs. We propose a novel in-context and generative pretraining approach, enabling few-shot adaptation during inference.
- Chapter 9 discusses the key limitations of our contributions and outlines directions for future research.

Chapter 2

Some notions on Partial Differential Equations

In this section, we present core concepts related to the resolution of partial differential equations. Our objective is to demonstrate how classical methods for solving PDEs operate and to provide intuition for readers without a background in physics. This section is not intended to be exhaustive, but aims to offer insights and perspectives that contextualize the contributions made in this thesis. Readers already familiar with partial differential equations may skip this chapter. For those seeking a deeper understanding, we recommend Olver (2007) as a comprehensive reference.

We begin by illustrating what a PDE is and how to solve it analytically through a simple example (Section 2.1). Next, we introduce fundamental concepts in the field (Section 2.2), followed by an overview of numerical solvers used to solve these equations (Section 2.3). Finally, we discuss dimensionality reduction and parameterization techniques (Section 2.4) that simplify the resolution process, highlighting the relevance and advantages of Deep Learning methods.

2.1 A first PDE example

Partial differential equations naturally arise from the mathematical formulation of physics first principles. These principles often involve the conservation of physical quantities, such as mass or momentum, across space and time. Here, we illustrate this derivation process through the heat equation, serving as our first PDE example. In a simplified setting, we then employ the method of separation of variables, an analytical technique, to obtain a solution to this equation.

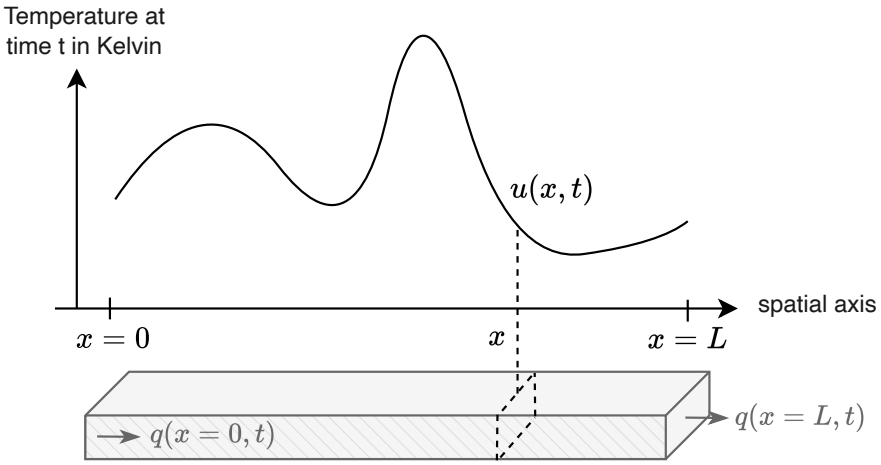


Figure 2.1: $u(x,t)$ is the temperature of the system at spatial coordinate x and time t expressed in Kelvin, while q is the heat flux (here in Watt, denoted W, as the system is 1-dimensional).

2.1.1 Heat equation derivation

The heat equation plays a fundamental role in engineering and science. It is a linear, homogeneous and second-order PDE, which makes it *canonical*, in the sense that it is possible to analytically derive solutions and study them. Therefore, it holds a particular place for better understanding partial differential equations and studying their resolution. This equation, first studied by Joseph Fourier himself, describes the evolution of the temperature in a solid material over time due to the conduction of heat.

Consider a rod of length L that we treat as one-dimensional, with mass density ρ (in kg/m^3), specific heat capacity c_p (in J/(kg·K)), and thermal conductivity κ (in W/(m·K)). The temperature $u(x,t)$ evolves as a function of the spatial coordinate $x \in [0, L]$ and time $t \in [0, T]$. For clarity, we define the interior of the domain as $\Omega = (0, L)$ and the boundary as $\partial\Omega = \{0\} \cup \{L\}$. The whole domain is $\bar{\Omega} = \Omega \cup \partial\Omega$.

To derive the governing equation for $u(x,t)$, we apply conservation principles to the system. In this scenario, where the medium is solid and no mass is transported, heat energy is the conserved quantity. The continuity equation reads as follows: *the rate of change of heat energy in time is equal to the instantaneous transfer of heat energy at the boundary of the system, plus the sources and sink of energy within the system.* Heat transfer in this one-dimensional setup occurs exclusively through conduction, as there is no convective heat transfer and we neglect radiative effects. Mathematically, this yields the following equation:

$$\frac{dQ}{dt} = (q(x = 0, t) - q(x = L, t)) + F(t) \tag{2.1}$$

where Q is the total heat energy of the system, q is the heat flux (J/s or W) that goes from high temperatures to low temperatures, and $F(t)$ is the source term within the domain Ω.

It is known from experimental observations (Cannon, 1984) that the total heat energy in the rod at time t is proportional to the temperature u, the mass density ρ, and the specific heat capacity c_p. Mathematically, this is expressed as:

$$Q(t) = \int_0^L \rho c_p u(x,t)\, dx, \tag{2.2}$$

Similarly, the source term can be expressed as an integral over the domain in function of a localized source term $f(x,t)$:

$$F(t) = \int_0^L f(x,t)\, dx, \tag{2.3}$$

On the other hand, the heat flux q, depends on the thermal conductivity κ of the material, and the temperature gradient $\frac{\partial u}{\partial x}$ according to Fourier's Law (Cannon, 1984). For a one-dimensional rod, the heat flux across the boundaries is given by:

$$q(x,t) = -\kappa \frac{\partial u(x,t)}{\partial x} \tag{2.4}$$

where the negative sign reflects the principle that heat flows from hotter to cooler regions. Using the fundamental theorem of calculus (analogous to Gauss's divergence theorem in higher dimensions), we can reformulate the following:

$$q(x = 0, t) - q(x = L, t) = -\int_0^L -\kappa \frac{\partial^2 u(x,t)}{\partial x^2} dx \tag{2.5}$$

Gathering all the pieces together, we can express everything in terms of domain integrals (instead of surface integrals) and we obtain the following integral equation:

$$\int_0^L \rho c_p \frac{\partial u}{\partial t}\, dx = \kappa \int_0^L \frac{\partial^2 u}{\partial x^2}\, dx + \int_0^L f(x,t)\, dx. \tag{2.6}$$

This equality must hold for any subdomain of the material, leading to the local form of the heat equation:

$$\rho c_p \frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2} + f(x,t) \tag{2.7}$$

For a homogeneous material where ρ, c_p, and κ are constant, the equation reads:

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2} + \frac{1}{\rho c_p} f(x,t), \tag{2.8}$$

where $\alpha = \frac{\kappa}{\rho c_p}$ is the thermal diffusivity, a parameter that quantifies the rate at which heat spreads through the material.

2.1.2 Initial boundary value problem formulation

To simplify the problem, we will neglect the source term ($f(x,t) = 0$) and focus on the homogeneous equation:

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2} \tag{2.9}$$

Once the physical problem has been translated into a mathematical equation, the next step is to determine how to find a temperature function $u(x,t)$ that satisfies the heat equation. Equally important is the question of whether a solution exists at all. For example, in the case of the 3D Navier-Stokes equations, it remains an open problem whether smooth solutions always exist (see the Millennium Prize problem of Clay Mathematics Institute (2000)). In contrast, the heat equation is relatively simple—it is linear, homogeneous, and second-order—which not only guarantees the existence of solutions but also often allows them to be expressed analytically.

The equation itself suggests that solutions must be at least once differentiable in time and twice differentiable in space, as these are necessary for the partial derivatives to be well-defined. To provide intuition that solutions to this problem exist, we can consider time-independent solutions, where $\frac{\partial u}{\partial t} = 0$. In this case, the heat equation reduces to the Laplace equation, which governs steady-state conditions. In one dimension, the Laplace equation simplifies further and can be solved by integrating twice over the spatial domain:

$$u(x,t) = \frac{\partial u}{\partial x}\Big|_{x=0} \cdot x + u(x=0).$$

As this example illustrates, the solution of the heat equation depends not only on the governing equation itself but also on the conditions imposed at the system's boundaries.

If we consider only Equation (2.9), the equation admits infinitely many solutions, making the problem inherently *ill-posed*. For instance, if u is a solution, then for any $\lambda \in \mathbb{R}^*$, the function λu is also a solution due to the equation's linearity. Thus, while the heat equation describes the fundamental dynamics of the system, additional constraints are required to fully determine a unique solution. To resolve this ambiguity, we introduce two essential elements:

- Initial conditions – These define the temperature state at $t = 0$, denoted $u^0(x)$.
- Boundary conditions – These account for external influences at the domain boundaries, which are not specified by the governing equation itself. Common boundary conditions include: (i) *Dirichlet* boundary conditions, which fix the temperature at the boundaries, e.g., $u(x = 0, t) = u_{|0}(t)$ and $u(x = L, t) = u_{|L}(t)$; (ii) *Neumann* boundary conditions, which specify the thermal flux at the boundaries, e.g., $q(x = 0, t) = 0$ and $q(x = L, t) = 0$, corresponding to insulated boundaries; (iii) *Periodic* boundary conditions, which enforce periodicity so that $u(x,t)\big|_{x=0} = u(x,t)\big|_{x=L}$, implying that spatial derivatives also match at these points.

With these conditions, we can ensure the problem has a unique solution (Muench, 2023) by formulating it as follows:

$$\begin{aligned}
&\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}, \quad \forall x \in \Omega, \\
&u(x, t=0) = u^0(x), \\
&\mathcal{B}(u)\big|_{\partial\Omega} = 0,
\end{aligned}$$

where $u^0(x)$ is a smooth function representing the initial state at $t = 0$, and $\mathcal{B}(T)$ is the boundary operator enforcing Neumann, Dirichlet, or periodic boundary conditions. These additional constraints resolve the ambiguity, ensuring the existence of a unique solution.

2.1.3 Analytical solutions

Now let us show, in a simple way, how to derive an analytical solution to the heat equation. For simplicity, we will consider periodic boundary conditions. The equation will be solved using the *separation of variables* principle (see Feldman (2007) for more details), which intuitively allows us to search for a solution $u(x,t)$ as a

product of a spatial function h and a temporal function g, i.e. $u(x,t) = h(x)g(t)$. By substituting $u(x,t) = h(x)g(t)$ into the heat equation, we obtain:

$$h(x)g'(t) = \alpha h''(x)g(t) \tag{2.10}$$

Rearranging the terms, and assuming $h(x)$ and $g(t)$ are always non-zero, we find:

$$\frac{h''(x)}{h(x)} = \frac{1}{\alpha}\frac{g'(t)}{g(t)} = -\lambda \tag{2.11}$$

where $\lambda \in \mathbb{R}$ is a separation constant. This follows from the observation that a function of x can only equal a function of t if both are constant. Thus, the separation of variables reduces the partial differential equation into a system of coupled ordinary differential equations:

$$h''(x) = -\lambda h(x),$$
$$g'(t) = -\alpha\lambda g(t).$$

The nature of the solutions for h depends on the sign of λ:

- Case 1: $\lambda = 0$
  The solution is $h(x) = ax + b$. However, periodic boundary conditions require $h(x + L) = h(x)$, leading to $aL + b = b$, which implies $a = 0$. Therefore, h is constant. This, in turn, makes g constant, contradicting the initial condition $u(x, t = 0) = u^0(x)$ which is generally not constant. Thus, $\lambda = 0$ yields no valid solutions.

- Case 2: $\lambda < 0$
  The solution for h is $h(x) = Ae^{\sqrt{-\lambda}x} + Be^{-\sqrt{-\lambda}x}$. For periodic boundary conditions, $h(x + L) = h(x)$ must hold. This condition forces $A = B = 0$, leaving only the trivial solution $u(x,t) = 0$.

- Case 3: $\lambda > 0$
  Let $\lambda = \mu^2$, where $\mu > 0$. The solution for h becomes: $h(x) = A\cos(\mu x) + B\sin(\mu x)$. Periodic boundary conditions require $h(x + L) = h(x)$, leading to $\sin(\mu L) = 0$. This implies $\mu = \frac{n\pi}{L}$ for some integer $n \geq 1$, giving $\lambda = \left(\frac{n\pi}{L}\right)^2$.

Denoting these eigenvalues by λ_n, we now solve for g. For each λ_n, the temporal equation becomes $g'(t) = -\alpha\lambda_n g(t)$, For which, the solution is an exponential decay $g(t) = e^{-\alpha\lambda_n t}$. By the superposition principle, the general solution is:

$$u(x,t) = \sum_{n=1}^{\infty}\left(A_n\cos\left(\frac{n\pi}{L}x\right) + B_n\sin\left(\frac{n\pi}{L}x\right)\right)e^{-\alpha\left(\frac{n\pi}{L}\right)^2 t}. \tag{2.12}$$

To determine A_n and B_n, we use the initial condition $u(x,0) = u^0(x)$. Projecting

this equation onto the Fourier basis functions $\cos\left(\frac{n\pi}{L}x\right)$ and $\sin\left(\frac{n\pi}{L}x\right)$, we find:

$$A_n = \frac{2}{L}\int_0^L u^0(x)\cos\left(\frac{n\pi}{L}x\right)dx, \quad B_n = \frac{2}{L}\int_0^L u^0(x)\sin\left(\frac{n\pi}{L}x\right)dx.$$

The solution illustrates how the heat equation smooths out an initial temperature distribution over time. Each Fourier mode (indexed by n) decays at a rate determined by its corresponding eigenvalue λ_n. Higher-frequency modes (with larger n) decay more rapidly due to the exponential factor $e^{-\alpha\left(\frac{n\pi}{L}\right)^2 t}$. As time increases, lower-frequency modes dominate the solution, resulting in an increasingly smoother temperature profile.

Notably, the separation of variables approach assumes that the solution can be expressed as a product of independent spatial and temporal components. However, by applying the superposition principle, the complete solution emerges as a sum of these separable solutions, each associated with a distinct mode.

This technique is just one of many tools available for analyzing and solving partial differential equations. For a more comprehensive exploration of analytical methods, such as the method of characteristics, readers may refer to an undergraduate-level textbook provided in Muench (2023).

2.2 ODEs and PDEs

Now that we understand how PDEs arise and that we have shown how they can be solved in an analytical way on a simple case, we will introduce some formalism and explain the different categories of PDEs that occur in scientific and engineering applications. Throughout this section, we will introduce some equations relevant in fluid mechanics, and which will motivate the use of numerical solvers.

2.2.1 Definitions

Before introducing formally partial differential equations in space and time, we provide the definition of a first-order ordinary differential equation (ODE) in time.

Definition 2.2.1. (First-order ODE) Let $u : \mathbb{R} \to \mathbb{R}^p$ be a differentiable function representing the state of a system at time t. A first-order ordinary differential equation is satisfied by u if there exists a continuous function $g : \mathbb{R}\times\mathbb{R}^p \to \mathbb{R}^p$ such that:

$$\forall t \in \mathbb{R}, \quad \frac{du}{dt} = g(t, u(t)), \tag{2.13}$$

where g maps the time scalar t and the state vector $u(t)$ to a vector in $\mathbb{R}^p$.

Note that in the case of an ODE, the state of the system u is a function of a single variable—the time variable. Consequently, no spatial derivatives need to be considered, unlike in the case of PDEs, where approximating spatial derivatives becomes essential. This means that if g is known, observing u at time t is theoretically sufficient to predict how the system will evolve over time. Notably, g can be nonlinear, as illustrated in the following example.

Example 2.2.1. The Lotka-Volterra equations (Lotka, 1925; Volterra, 1928) form a system of first-order nonlinear differential equations that model the interaction between two populations: prey and predators. The prey population grows in the absence of predators but declines due to predation, while the predator population decreases without prey but thrives when prey are abundant. The system is described by:

$$\frac{dx}{dt} = ax - bxy, \quad \frac{dy}{dt} = cxy - dy,$$

where $x(t)$ and $y(t)$ represent the prey and predator populations, respectively. The parameters a, b, c, and d correspond to the intrinsic reproduction rate of the prey, the rate of prey mortality due to predation, the reproduction rate of predators proportional to the prey consumed, and the intrinsic mortality rate of predators.

As in the introductory PDE example, we are interested in finding a solution $u(t) = (x(t), y(t))$ that satisfies the ODE for a given particular initial condition u^0. This is called an *initial value problem*, and we provide its definition below.

Definition 2.2.2. (Initial Value Problem) Given a continuous function $g : \mathbb{R} \times \mathbb{R}^p \to \mathbb{R}^p$ and an initial state u^0, solving an initial value problem (IVP) for a first-order ordinary differential equation involves finding a differentiable function $u : \mathbb{R} \to \mathbb{R}^p$ that satisfies the ODE:

$$\frac{du}{dt} = g(t, u(t)),$$

and the initial condition:

$$u(0) = u^0,$$

The solution $u(t)$ describes the system's trajectory starting from u^0 at $t = 0$.

Under some circumstances, there is an existence and uniqueness guarantee of the solution to this initial value problem. The proofs of existence and uniqueness can be found in Peano (1890) for example.

Example 2.2.2. In the general case, it is not possible to solve the IVP analytically for the Lotka-Volterra equations. However, as we will show in the section on numerical schemes, it is possible to discretize the dynamics and use time-integration schemes based on e.g. Euler's or Runge-Kutta methods (Runge (1895); Kutta (1901)). We show an example of a solution in Figure 2.2 with a Runge-Kutta solver. Notice the periodic patterns of the approximated solutions, which is characteristic of the system's solution.

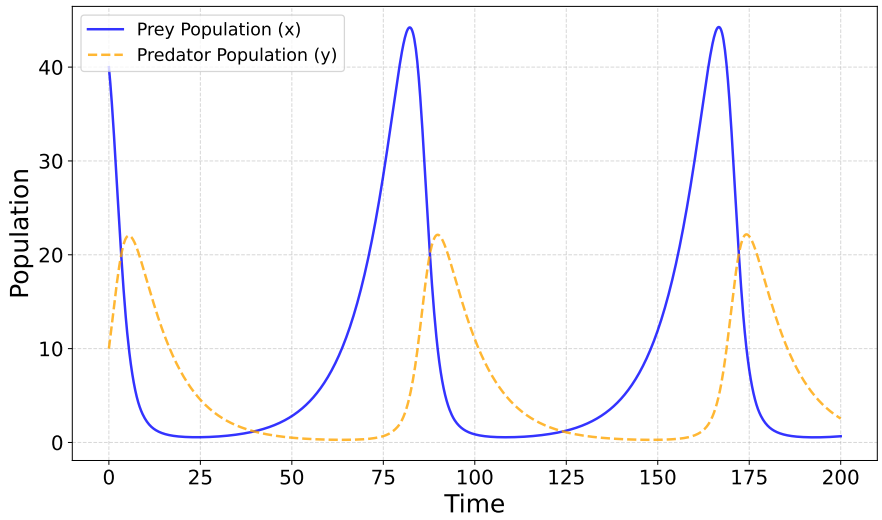


Figure 2.2: Discretized solution of the LV equations. Starting with $(x(0) = 40, y(0) = 10)$ and with parameters $a = d = 0.1$ and $b = 0.02$ and $c = 0.01$. We obtain the following solution on $[0, 200]$ with an RK45 scheme.

We now turn to generally introduce the notion of PDE. The main difference between an ODE and a PDE is that the equation involves several partial derivatives (at least two) as the state of interest is a function of multiple variables, mainly space and time.

Definition 2.2.3. (Partial Differential Equation in space and time) Let Ω be an open set included in $\mathbb{R}^d$, where d is the dimension of the system. Let $u : \Omega \times \mathbb{R} \to \mathbb{R}^p$ be a smooth function representing the state of a system at spatial location $x = (x_1, \cdots, x_d)$ and time t. A partial differential equation in space and time is satisfied by u if there exists a continuous function g such that:

$$\forall x, t \in \Omega \times \mathbb{R}, \quad \frac{\partial u}{\partial t} = g(\mu, x, t, f(x,t), u(t), \nabla u, \nabla \cdot u, \nabla \times u, \nabla^2 u, \cdots) \quad (2.14)$$

where g is now a function of u and its spatial derivatives, but also of coefficient

parameters μ and of an external forcing function $f(x,t)$.

Note that here, we use the ∇ notation to represent the operators involving spatial derivatives. Specifically, ∇u, $\nabla \cdot u$, $\nabla \times u$, and ∇^2 denote the gradient, divergence, curl and laplacian of the field u, respectively. We have already provided a PDE example in Section 2.1 with the Heat equation, so let us give another example related to fluid dynamics.

Example 2.2.3. Burgers' equation is a simplification of the Navier-Stokes equation (by neglecting the gradient of the pressure) used for representing convective and diffusive flows. It is a non-linear equation commonly studied in one-dimension ($d = 1$):

$$\frac{\partial u}{\partial t} = \nu \frac{\partial^2 u}{\partial x^2} - u \frac{\partial u}{\partial x} \tag{2.15}$$

or equivalently using the operators from Equation (2.14):

$$\frac{\partial u}{\partial t} = \nu \nabla^2 u - (u \cdot \nabla) u, \tag{2.16}$$

where $u(x,t)$ represents the fluid's velocity, and ν is the viscosity. The terms $u\frac{\partial u}{\partial x}$ and $\nu\frac{\partial^2 u}{\partial x^2}$ are the convective (advection) and diffusive effects on the dynamics. The Burgers' equation illustrates the formation of shock waves (when $\nu \to 0$).

As discussed in Section 2.1, specifying the initial and boundary conditions is essential to fully characterize the system, a formulation known as an *initial boundary value problem*. Additionally, multiple conservation laws may be required to provide independent constraints on the system. A system is considered *closed*, or fully characterized, when there are, in theory, a sufficient number of equations, along with appropriate initial and boundary conditions, to uniquely determine the solution. Furthermore, the initial boundary value problem is considered *well-posed* if it satisfies three criteria: (1) a solution exists, (2) the solution is unique, and (3) the solution depends continuously on the input parameters (i.e. it is stable).

Definition 2.2.4. (Initial Boundary Value Problem or IBVP). The state u satisfies an initial boundary value problem associated with Equation (2.14) if:

$$\begin{aligned} \frac{\partial u}{\partial t} &= g(\mu, x, t, f(x,t), u, \nabla u, \nabla^2 u, \dots), && \forall (x,t) \in \Omega \times \mathbb{R}, \\ u(x, t_0) &= u^0(x), && \forall x \in \Omega, \\ \mathcal{B}(u, \nabla u, x, t) &= 0, && \forall (x,t) \in \partial\Omega \times \mathbb{R}, \end{aligned} \tag{2.17}$$

where $\mathcal{B}$ is a boundary operator involving u and its spatial derivatives on $\partial\Omega$ (e.g., Dirichlet or Neumann conditions), and $u^0(x)$ is the initial condition.

Example 2.2.4. Even though Burgers' equation is non-linear, it remains possible to analyze its solutions. For $\nu > 0$, analytical solutions can be obtained using the Cole-Hopf transformation (Hopf, 1950), which transforms the equation into a heat equation with a change of variable. In the inviscid case ($\nu = 0$), the solution can be derived and analyzed using the method of characteristics, an approach suitable for transport-type phenomena. Notably, for specific affine initial conditions, it can be shown that a time step t^* exists, called the *breaking time*, at which point in time a shock forms and the solution becomes singular (Cameron, 2016).

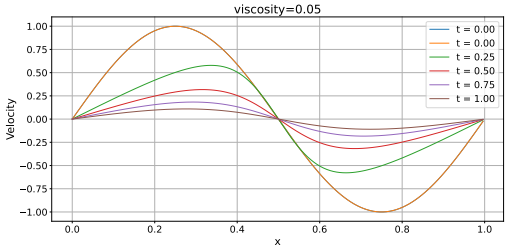
To solve our first initial boundary value problem, we use a periodic initial condition $u^0 = \sin(2\pi x)$ along with periodic boundary conditions. We illustrate how the viscosity parameter ν influences the solution obtained with a numerical solver in Figure 2.3. As ν decreases, we observe the formation of a shock in the middle of the domain, and numerical errors become apparent due to the suboptimal choice of the numerical solver. For higher values of ν, the amplitude of the signal decays more rapidly towards zero, as there is no external forcing term adding energy to the system. This demonstrates that the diffusion term in Burgers' equation has a dissipative effect on the system, gradually reducing its kinetic energy. However, in all cases, the global momentum is conserved.

2.2.2 Fundamental properties of PDEs

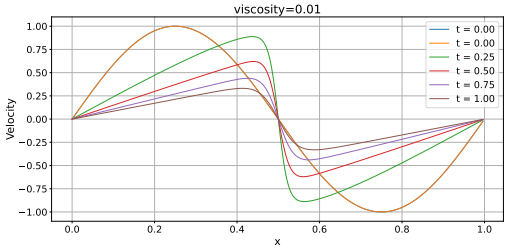
Definition 2.2.5. (linearity) A PDE is linear if g can be expressed as a linear combination of the dependent variable u and its derivatives $\nabla u, \nabla^2 u$, etc.

Linear equations are generally easier to solve than nonlinear ones, and can be better understood analytically (Olver, 2007). They are often more predictable whereas nonlinear equations can include bifurcations or chaotic phenomenon. From a physical point of view, homogeneous equations often represent systems without external forcing.

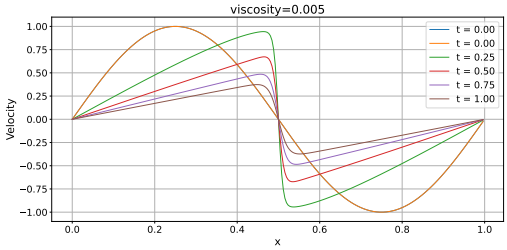
Example 2.2.5. The heat equation is an example of a linear equation. Based on Equation (2.9), we have $g(\alpha, x, t, f(x,t), u, \nabla u, \nabla \cdot u, \nabla \times u, \nabla^2 u, \dots) = \alpha \nabla^2 u + f(x,t)$.



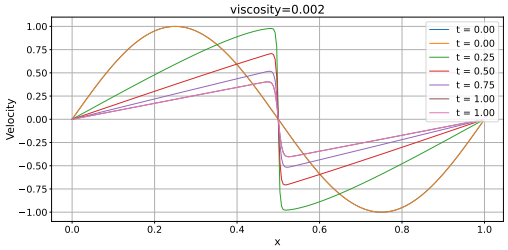
(a) $\nu = 0.05$



(b) $\nu = 0.01$



(c) $\nu = 0.005$



(d) $\nu = 0.002$

Figure 2.3: Examples of a solution of the Burgers' equation for various viscosity value. In all cases, the initial condition is set to $u_0(x) = \sin(2\pi x)$ and we fix periodic boundary conditions. For the numerical solver, we employed a finite difference scheme of order 4 to approximate the spatial derivatives and an explicit runge kutta of order 4 for time-integration. See Section 2.3 for further details.

Definition 2.2.6. (homogeneity) A PDE is homogeneous if $f(x,t) = 0$; otherwise, it is inhomogeneous.

An important property of homogeneous linear PDEs is that if u_1 and u_2 are solutions, then by linearity $u_1 + u_2$ is also a solution. This is called the superposition principle, and this is what we used to express the general solution of the heat equation in Section 2.1.

Example 2.2.6. Laplace's equation is the homogeneous version of the Poisson equation:

$$\nabla^2 u(x) = f(x) \quad \forall x \in \Omega \quad \text{(Poisson)}$$
$$\nabla^2 u(x) = 0 \quad \forall x \in \Omega \quad \text{(Laplace)}$$

The Poisson equation is widely used in various scientific applications, notably for computing pressure fields in the solution of incompressible Navier-Stokes equations (Ferziger and Peric, 1996).

Definition 2.2.7. (order) The order of a PDE is determined by the highest derivative of the variable u in Equation (2.14).

In this thesis, we will primarily focus on first-order and second-order equations, with a few exceptions. Notably, the Korteweg-de Vries equation (see, e.g. Miura et al. (1968)) and the Kuramoto-Sivashinsky equation (see e.g. Kuramoto (1978)) are well-studied nonlinear equations of third and fourth order, respectively. As the order of an equation increases, its numerical solution becomes more complex and challenging.

Example 2.2.7. Advection equation represents the transport of a scalar quantity with constant speed β. It is a first-order PDE: $\frac{\partial u}{\partial t} + \beta \frac{\partial u}{\partial x} = 0$. The wave equation describes the propagation of a wave traveling at a constant celerity c. It is a second-order partial differential equation (PDE) $\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}$. Here, the second derivative with respect to time appears, which differs from our formulation in Equation (2.14). However, we can rewrite this as a system of first-order-in-time equations by introducing the extended state vector $x = \left(u, \frac{\partial u}{\partial t}\right)$.

Linear, homogeneous, second-order partial differential equations can be broadly classified into three categories: elliptic, parabolic, and hyperbolic equations (Olver, 2007). While these categories have precise mathematical definitions, we will describe them using physical intuition and properties, as these insights are also useful for understanding non-linear equations. An exhaustive description of linear partial differential equations can be found in Polyanin (2001).

Hyperbolic equations govern systems where information propagate at *finite speeds*, such as waves. Therefore, a defining characteristic of hyperbolic systems is that information is confined to a specific region, meaning it cannot travel faster than the wave celerity in the medium. This ensures causality, maintaining a clear relationship between different points in space and time. Examples include shock waves generated by supersonic aircraft and sound waves propagating through a compressible medium like air.

Parabolic equations describe systems where information *diffuses* over time and space in a smooth and gradual manner, without abrupt changes or shocks. These equations commonly model diffusion processes, such as heat conduction. A key property of parabolic equations is that information typically propagates in a single direction—often forward in time. For instance, once heat is transferred into a material, it spreads continuously from hot to cooler regions, and does not transfer backward.

Elliptic equations, in contrast, describe systems in a *steady-state* or equilibrium, where there is no explicit time dependence. In these systems, the state at any point is determined by its spatial surroundings, where every point influence each other. A well-known example is the Poisson equation, often used to calculate the electrostatic potential around a static charge or the gravitational potential within a given mass distribution.

2.2.3 Non-linear equations

A PDE is *nonlinear* if u or its derivatives appear with nonlinear terms. This can include for example the convective term $(u \cdot \nabla)u$ for fluid flows, or the reaction term $f(u) = u(1-u)$ in chemical reactions. Nonlinearity significantly complicates analysis and the resolutions, and as a result non-linear equations often cannot be solved explicitly and require approximations or numerical methods.

Example 2.2.8. The compressible Navier-Stokes equations describe the motion of a viscous, compressible fluid, where the density can vary in space and time. These equations are derived from the principles of conservation of mass, conservation of momentum, and conservation of energy (in case of heat transfer):

$$\begin{aligned} \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho u) &= 0 \quad \text{(Continuity equation: Conservation of mass)} \\ \frac{\partial \rho u}{\partial t} + (u \cdot \nabla)u &= -\frac{1}{\rho}\nabla p + \nu \nabla^2 u + f \quad \text{(Conservation of momentum)} \\ \frac{\partial E}{\partial t} + \nabla \cdot (u(E+p)) &= \nu \left(\nabla u\right)^2 + q \quad \text{(Conservation of energy)} \end{aligned} \tag{2.18}$$

Where u is the fluid velocity, ρ is the density of the fluid, p the pressure, f a

forcing term, E the energy of the system per unit mass, q a heat flux and ν is the kinematic viscosity. The hypothesis of a compressible fluid is often verified for gas fluids with high velocity V. A general rule of thumb indicates that a gas flow should be considered compressible for mach numbers greater than 0.3 (Ferziger and Peric, 1996). The mach number is a dimensionless number $M = \frac{V}{c}$ where c is the speed of sound in the medium.

However, for liquids and gas fluids with $M < 0.3$ it is in general better to assume that the fluid is incompressible and has a constant density ρ. This amounts to have $\frac{\partial \rho}{\partial t} = 0$, and the conservation of mass therefore yields $\nabla \cdot u = 0$, i.e. the velocity has a zero divergence. Furthermore, if we consider the fluid to be isothermal, this yields the incompressible Navier-Stokes equation:

$$\begin{aligned} \nabla \cdot u &= 0 \quad \text{(Continuity equation)} \\ \frac{\partial u}{\partial t} + (u \cdot \nabla)u &= -\nabla p + \nu \nabla^2 u + f \quad \text{(Conservation of momentum)} \end{aligned} \tag{2.19}$$

In this thesis, we will examine both liquid and gas flows with Mach number smaller than 1. Flows with Mach numbers greater than 1 are referred to as supersonic, and are characterized with shock waves.

The Reynolds number is a key parameter used to characterize the flow regime of a fluid. It helps to classify a flow as *laminar* or *turbulent*, with lower Reynolds numbers indicating smooth flows and higher numbers indicating chaotic, irregular flows. The Reynolds number Re is a dimensionless quantity that compares inertial forces to viscous forces in a fluid flow, defined as $\mathrm{Re} = \frac{uL}{\nu}$, where u is the velocity, L is a characteristic length, and ν is the kinematic viscosity. As the Reynolds number increases, the flow becomes more dominated by inertial forces, leading to turbulence. As a result, a low viscosity (small ν) increases the Reynolds number, making the fluid more turbulent.

Another distinctive feature of nonlinear equations is that small changes in the initial conditions of the system can lead to drastically different solutions. Therefore nonlinear systems can be chaotic. This is famously known as the butterfly effect (Lorenz, 1963), where small changes in the initial conditions of the atmospheric system could lead to vastly different weather outcomes. This contrasts sharply with linear equations, where the solutions typically change smoothly in response to changes in the input.

Example 2.2.9. A simple example of a chaotic Partial Differential Equation (PDE) is the Kuramoto-Sivashinsky (KS) equation. The KS equation is a nonlinear PDE often used to model spatiotemporal chaos in a variety of systems such as flame fronts. The Kuramoto-Sivashinsky equation for a scalar field u in one dimension is:

$$\frac{\partial u}{\partial t} = -u\frac{\partial u}{\partial x} - \frac{\partial^2 u}{\partial x^2} - \frac{\partial^4 u}{\partial x^4}$$

Chaotic phenomenon are much harder to reproduce numerically as small perturbations lead to drastically different solutions. We will focus on non-chaotic equations in this thesis.

2.3 Numerical Solvers

As discussed in the previous section, many non-linear equations lack analytical solutions, and even linear equations can become difficult to solve with complex boundary conditions. To address these challenges, numerical methods provide powerful tools for obtaining approximate solutions to partial differential equations by discretizing the problem.

In this section, we will introduce key numerical techniques for discretizing and solving PDEs, considering both spatial and temporal dimensions. Widely used methods such as finite differences, finite elements, and various time integration schemes each offer unique advantages and trade-offs.

We begin with the finite difference method (FDM), a straightforward approach that approximates spatial derivatives by discretizing the domain on a structured grid. Next, we explore the finite element method (FEM), which leverages the weak formulation of the PDE to handle complex geometries and boundary conditions with greater flexibility. Finally, we discuss time integration schemes, covering explicit methods like Euler and Runge-Kutta, as well as implicit methods and iterative solvers for stable time-stepping in more demanding scenarios.

2.3.1 Finite differences

The finite difference method is a numerical technique used to approximate derivatives in PDEs. It expresses the derivatives of a function u, which depends on both space x and time t, as differences between values of u at discrete grid points $x_i = i\Delta x$. To approximate the derivative of u at a spatial grid point x_i, we can consider its Taylor series expansion around x_i. For a small spatial increment Δx, the function at $x_{i+1} = x_i + \Delta x$ can be written as:

$$u(x_{i+1}, t) = u(x_i, t) + \Delta x\frac{\partial u}{\partial x}(x_i, t) + \frac{\Delta x^2}{2!}\frac{\partial^2 u}{\partial x^2}(x_i, t) + \frac{\Delta x^3}{3!}\frac{\partial^3 u}{\partial x^3}(x_i, t) + O(\Delta x^4). \quad (2.20)$$

Similarly, at $x_{i-1} = x_i - \Delta x$, the expansion is:

$$u(x_{i-1}, t) = u(x_i, t) - \Delta x\frac{\partial u}{\partial x}(x_i, t) + \frac{\Delta x^2}{2!}\frac{\partial^2 u}{\partial x^2}(x_i, t) - \frac{\Delta x^3}{3!}\frac{\partial^3 u}{\partial x^3}(x_i, t) + O(\Delta x^4). \quad (2.21)$$

These expansions can be combined to derive various finite difference schemes. In Equation (2.20), subtracting $u(x_i, t)$ from $u(x_{i+1}, t)$, we obtain the forward difference scheme (FDS):

$$\frac{u(x_{i+1}, t) - u(x_i, t)}{\Delta x} = \frac{\partial u}{\partial x}(x_i, t) + \frac{\Delta x}{2!}\frac{\partial^2 u}{\partial x^2}(x_i, t) + \frac{\Delta x^2}{3!}\frac{\partial^3 u}{\partial x^3}(x_i, t) + O(\Delta x^3)$$

If we rearrange the terms of the Taylor expansion, we obtain:

$$\frac{\partial u}{\partial x}(x_i, t) = \underbrace{\frac{u(x_{i+1}, t) - u(x_i, t)}{\Delta x}}_{\text{Discrete approximation}} \underbrace{- \frac{\Delta x}{2!}\frac{\partial^2 u}{\partial x^2}(x_i, t) - \frac{\Delta x^2}{3!}\frac{\partial^3 u}{\partial x^3}(x_i, t) + O(\Delta x^3)}_{\text{Truncation error}}$$

where we emphasize the *truncation error* resulting from this finite difference approximation, which arises from omitting higher-order terms in the Taylor expansion. The leading term in this truncation error is of the order Δx, indicating that this scheme is *first-order accurate*. More generally, a scheme of order p has a leading truncation error term of the order of $O(\Delta x^p)$.

Similarly, we obtain the backward difference scheme (BDS) from Equation (2.21):

$$\frac{u(x_i, t) - u(x_{i-1}, t)}{\Delta x} = \frac{\partial u}{\partial x}(x_i, t) - \frac{\Delta x}{2!}\frac{\partial^2 u}{\partial x^2}(x_i, t) + \frac{\Delta x^2}{3!}\frac{\partial^3 u}{\partial x^3}(x_i, t) + O(\Delta x^3)$$

This is also a first-order scheme with an $O(\Delta x)$ error. Finally, subtracting Equation (2.21) from Equation (2.20) and dividing by $2\Delta x$ yields the central difference scheme (CDS), which is second-order accurate:

$$\frac{u(x_{i+1}, t) - u(x_{i-1}, t)}{2\Delta x} = \frac{\partial u}{\partial x}(x_i, t) + \frac{\Delta x^2}{3!}\frac{\partial^3 u}{\partial x^3}(x_i, t) + O(\Delta x^4)$$

We can also combine equations Equation (2.20) and Equation (2.21) to obtain an approximation of the second order derivative:

$$\frac{u(x_{i+1}, t) - 2u(x_i, t) + u(x_{i-1}, t)}{\Delta x^2} = \frac{\partial^2 u}{\partial x^2}(x_i, t) + O(\Delta x^2)$$

The order of a scheme has direct implications for the accuracy and computational costs of numerical solvers. Higher-order schemes provide more accurate approximations for the same grid spacing. For this reason, a central difference scheme, which has a second-order local truncation error, is generally preferred over forward or backward difference schemes, which are first-order accurate.

To improve accuracy, higher-order finite difference schemes can be used by incorporating additional grid points into the *stencil* and determining coefficients that minimize the truncation error. The stencil defines the local points that are used to compute the approximate derivatives. Here, for the central difference scheme

to approximate the first derivative, we used the stencil $(-1, 1)$ with coefficients $\frac{1}{2\Delta x}(-1, 1)$, while we used the stencil $(-1, 0, 1)$ with coefficients $\frac{1}{\Delta x^2}(1, -2, 1)$ to approximate the second-order derivative.

Using Taylor expansions at points such as x_{i+2} and x_{i-2}, or even further, higher-order schemes can be derived. However, this increased accuracy comes at the cost of greater computational complexity. For example:

$$\frac{\partial u}{\partial x}(x_i, t) \approx \frac{-u(x_{i+2}, t) + 8u(x_{i+1}, t) - 8u(x_{i-1}, t) + u(x_{i-2}, t)}{12\Delta x},$$

is a fourth-order finite difference scheme with stencil $(-2, -1, 0, 1, 2)$ and coefficients $\frac{1}{12\Delta x}(-1, 8, 0, -8, 1)$.

Example 2.3.1. Returning to our beloved heat equation from Section 2.1:

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$

we now discretize the spatial domain into N equally spaced nodes $x_1, x_2, \ldots, x_N$ with a uniform spacing Δx. Using a second-order central difference scheme to approximate the second spatial derivative at time t, we have for each node x_i:

$$\frac{\partial u}{\partial t}(x_i, t) = \alpha \frac{u(x_{i+1}, t) - 2u(x_i, t) + u(x_{i-1}, t)}{\Delta x^2}$$

This transforms the continuous PDE into a discrete system of coupled ordinary differential equations. For all nodes $i = 1, \ldots, N$, we can write the discrete equations compactly as a linear system:

$$\frac{d\mathbf{u}}{dt} = \mathbf{A}\mathbf{u}(t), \tag{2.22}$$

where $\mathbf{u}(t) = \begin{bmatrix} u(x_1, t) & u(x_2, t) & \cdots & u(x_N, t) \end{bmatrix}^\top$ is the state vector at time t representing the temperature at each spatial node. The matrix $\mathbf{A}$ encodes the finite difference discretization of the second derivative and is given by:

$$\mathbf{A} = \frac{\alpha}{\Delta x^2} \begin{bmatrix} -2 & 1 & 0 & \cdots & 0 & 1 \\ 1 & -2 & 1 & \cdots & 0 & 0 \\ 0 & 1 & -2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 0 & 0 & \cdots & 1 & -2 \end{bmatrix}.$$

The first and last entries of the matrix (top-right and bottom-left corners) enforce periodic boundary conditions, where the first node x_1 is connected to the last node x_N, and vice versa. For other types of boundary conditions (e.g.,

Dirichlet or Neumann), these entries would need to be modified accordingly.

This choice of discretization and approximation reduces the resolution of a PDE problem into that of a system of linear ODE. The matrix $\mathbf{A}$ encapsulates the spatial discretization, and the evolution of $\mathbf{u}(t)$ over time is governed by this matrix. The numerical solution can then proceed using time integration methods such as Euler, Runge-Kutta, or implicit solvers, depending on the desired stability and accuracy properties, that we will present in Section 2.3.3.

Taylor series expansion is not the only method to approximate derivatives. Local polynomial fitting uses neighboring grid points to construct a polynomial approximation of u. The derivative is then computed from the polynomial, enabling higher-order accuracy (Brabanter et al., 2013). Padé approximants express derivatives as ratios of polynomials, achieving high accuracy with relatively few grid points. This method is particularly useful for stiff problems or where high precision is required (Baker, 1975).

In practice, for modeling spatiotemporal dynamics with deep learning models, we will also separate space and time as is done with this example. We discretize the solution in space and advance the system forward in time with time-stepping. This autoregressive setting will be the foundation of the contributions that we present in this thesis.

2.3.2 Finite elements

Finite difference methods rely on structured grids and regular discretizations, making them less suitable for handling complex geometries, non-uniform meshing, and general boundary conditions. These limitations stem from the reliance of FDM on approximating derivatives at discrete grid points, which becomes challenging to apply effectively on irregular domains or domains with intricate boundary shapes.

In contrast, the finite element method (FEM) is a versatile numerical technique for solving partial differential equations, and is particularly well-suited for problems involving complex geometries, irregular domains, and non-uniform meshes. The method starts from the so-called *weak* formulation of the PDE, which reformulates the problem in terms of integrals rather than derivatives. This reformulation inherently provides flexibility, allowing FEM to yield potentially discontinuous solutions.

Let us revisit the PDE formulation of Equation (2.14). A function u is said to satisfy the *strong formulation* of the PDE if

$$\forall t \in \mathbb{R},\ \forall x \in \Omega, \quad \frac{\partial u}{\partial t} = g(u) + f(x,t),$$

where g represents an operator applied to u and its derivatives, and f is the forcing term. In the *weak formulation*, we test the PDE against a function $v \in V$, where V is a suitable function space defined on Ω with sufficient regularity. We say that u

satisfies the weak formulation of Equation (2.14) if

$$\forall t \in \mathbb{R},\, \forall v \in V, \quad \int_\Omega \frac{\partial u}{\partial t} v \, dx - \int_\Omega g(u) v \, dx = \int_\Omega f v \, dx.$$

Using the notation $\langle f, g \rangle_\Omega = \int_\Omega f g \, dx$, this can be written more compactly as

$$\forall t \in \mathbb{R},\, \forall v \in V, \quad \langle \frac{\partial u}{\partial t} - g(u), v \rangle_\Omega = \langle f, v \rangle_\Omega.$$

The weak formulation shifts the problem from finding u, a solution satisfying the PDE pointwise, to finding u in a weaker sense. Specifically, u is required to satisfy the integral equation for every test function v. Importantly, every solution to the strong formulation is also a solution to the weak formulation, but the inverse is not necessarily true, as the weak formulation admits solutions with lower regularity, such as functions with discontinuous derivatives.

In FEM, the solution u and the test functions v are approximated using finite-dimensional function spaces, typically spanned by piecewise polynomials. The domain Ω is divided into smaller subdomains, called elements Ω_i, over which the functions are simple, such as linear or quadratic polynomials. This approach provides the flexibility to handle arbitrary geometries, non-uniform meshing, and general boundary conditions, making FEM a powerful tool for solving PDEs in diverse applications. More details can be found for example in Johnson (1992).

Example 2.3.2. Let us consider the heat equation again, this time with a forcing term f:

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2} + f,$$

The corresponding weak formulation is:

$$\forall t \in \mathbb{R}, \forall v \in V, \quad \int_\Omega \frac{\partial u}{\partial t} v(x) \, dx - \int_\Omega \alpha \frac{\partial^2 u}{\partial x^2} v(x) \, dx = \int_\Omega f(x,t) v(x) \, dx,$$

where V is a suitable space of test functions, such as $V = H^1(\Omega)$, the Sobolev space of functions with square-integrable derivatives. We also impose periodicity constraints on v, i.e. $v(0) = v(L)$ and $\frac{\partial v}{\partial x}(0) = \frac{\partial v}{\partial x}(L)$. Using the divergence theorem, we integrate by parts to reduce the order of the derivatives:

$$\int_\Omega \frac{\partial u}{\partial t} v(x) \, dx - \alpha \left(\int_{\partial\Omega} \frac{\partial u}{\partial x} v(x) \, dx - \int_\Omega \frac{\partial u}{\partial x} \frac{\partial v}{\partial x} \, dx \right) = \int_\Omega f(x,t) v(x) \, dx.$$

Since u and v are periodic over the domain, the boundary term vanishes, and the weak formulation simplifies to:

$$\forall t \in \mathbb{R}, \forall v \in V, \quad \int_\Omega \frac{\partial u}{\partial t} v(x)\, dx + \alpha \int_\Omega \frac{\partial u}{\partial x}\frac{\partial v}{\partial x}\, dx = \int_\Omega f(x,t) v(x)\, dx. \quad (2.23)$$

The solution u must satisfy Equation (2.23) for every function $v \in V$.
To discretize in space, we partition $\Omega = [0, L]$ into N subintervals:

$$x_0 = 0, \quad x_1 = \Delta x, \quad x_2 = 2\,\Delta x, \quad \ldots, \quad x_N = L,$$

where $\Delta x = \frac{L}{N}$. On each subinterval $[x_i, x_{i+1}]$, we define piecewise linear basis functions $\{\phi_i\}_{j=0}^{N}$. A typical function ϕ_i is given by

$$\phi_i(x) = \begin{cases} \dfrac{x - x_{i-1}}{\Delta x}, & x \in [x_{i-1}, x_i], \\ 1 - \dfrac{x - x_i}{\Delta x}, & x \in [x_i, x_{i+1}], \\ 0, & \text{otherwise.} \end{cases}$$

Since we want u to be periodic, we identify the left boundary $x_0 = 0$ with the right boundary $x_N = L$. Equivalently, we treat node 0 and node N as the same physical point, reducing the number of independent degrees of freedom by one. Concretely, let

$$u_\Delta(x,t) \;=\; \sum_{i=0}^{N-1} T_i(t)\; \phi_i(x),$$

where we implicitly enforce $T_0(t) = T_N(t)$. We substitute u_Δ into Equation (2.23) and choose $v(x) = \phi_j(x)$ for $j = 0, \ldots, N-1$. This yields

$$\sum_{i=0}^{N-1} \frac{dT_i(t)}{dt} \int_\Omega \phi_i(x)\phi_j(x)\, dx + \alpha \sum_{i=0}^{N-1} T_i(t) \int_\Omega \frac{\partial \phi_i}{\partial x}\frac{\partial \phi_j}{\partial x}\, dx = \int_\Omega f(x,t)\phi_j(x)\, dx.$$

Defining the mass matrix $\mathbf{M}$, stiffness matrix $\mathbf{K}$, and load vector $\mathbf{F}$:

$$\begin{aligned} M_{ij} &= \int_0^L \phi_i(x)\, \phi_j(x)\, dx, \\ K_{ij} &= \int_0^L \frac{\partial \phi_i}{\partial x}(x)\, \frac{\partial \phi_j}{\partial x}(x)\, dx, \\ F_j(t) &= \int_0^L f(x,t)\, \phi_j(x)\, dx. \end{aligned}$$

Hence, for each $j \in \{0, \dots, N-1\}$, we have:

$$\sum_{i=0}^{N-1} \frac{dT_i}{dt} M_{ij} \; + \; \alpha \sum_{i=0}^{N-1} T_i \, K_{ij} \; = \; F_j(t).$$

In matrix form, this becomes:

$$\mathbf{M} \frac{d\mathbf{T}}{dt} \; + \; \alpha \, \mathbf{K} \, \mathbf{T} \; = \; \mathbf{F}(t), \tag{2.24}$$

where $\mathbf{T}(t) = \big(T_0(t), T_1(t), \dots, T_{N-1}(t)\big)^\top$. Solving this system with appropriate time integration yields the approximate solution.

2.3.3 Time integration

Once the discretization through finite differences or finite elements has transformed a partial differential equation (PDE) into a high-dimensional ordinary differential equation (ODE), we can apply standard time integration methods from the ODE literature. This approach—often referred to as the *method of lines*—leads us to consider the generic formulation from Equation (2.13) for the remainder of this section.

Time integration methods are essential for solving partial differential equations (PDEs) by advancing solutions in time, given initial or boundary conditions. These methods can be categorized as explicit, implicit, or iterative/direct, depending on their approach to time evolution.

Explicit methods compute the solution at the next time step using information from the current and previous steps. The simplest explicit scheme is the Forward Euler method, given by

$$\mathbf{u}(t + \Delta t) = \mathbf{u}(t) + \Delta t \cdot g(\mathbf{u}(t), t),$$

where $u(t + \Delta t)$ represents the solution at time $t + \Delta t$, and $g(u(t), t)$ is the time derivative evaluated at the current time step. This method is first-order accurate but conditionally stable for solving PDEs. Indeed, in practice $g(u(t), t)$ approximates spatial derivatives and depends on the spatial discretization parameter Δx, therefore, the time step Δt must satisfy the Courant-Friedrichs-Lewy (CFL) condition for stability. For higher accuracy, Runge-Kutta methods, such as the fourth-order Runge-Kutta (RK4), are often employed. The RK4 method computes the solution at intermediate stages k_i and advances the solution as

$$\mathbf{u}(t + \Delta t) = \mathbf{u}(t) + \frac{\Delta t}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4),$$

where the intermediate evaluations are

$$\mathbf{k}_1 = g(\mathbf{u}(t), t), \quad \mathbf{k}_2 = g\left(\mathbf{u}(t) + \frac{\Delta t}{2}\mathbf{k}_1, t + \frac{\Delta t}{2}\right),$$

$$\mathbf{k}_3 = g\left(\mathbf{u}(t) + \frac{\Delta t}{2}\mathbf{k}_2, t + \frac{\Delta t}{2}\right), \quad \mathbf{k}_4 = g\left(\mathbf{u}(t) + \Delta t\,\mathbf{k}_3, t + \Delta t\right).$$

Explicit methods are efficient for smooth, non-stiff problems but require small time steps for stability.

On the other hand, *implicit methods* compute the solution at the next time step by solving equations that involve the future state $u(t + \Delta t)$. The Backward Euler method is a simple implicit scheme:

$$\mathbf{u}(t + \Delta t) = \mathbf{u}(t) + \Delta t \cdot g(\mathbf{u}(t + \Delta t), t + \Delta t),$$

Because $\mathbf{u}(t+\Delta t)$ appears on both sides, a nonlinear (or linearized) system must be solved at every time step. While Backward Euler is unconditionally stable for linear problems, using too large a time step can still lead to large errors. To achieve higher accuracy, the Crank-Nicolson method is often used, which is second-order accurate:

$$\mathbf{u}(t + \Delta t) = \mathbf{u}(t) + \frac{\Delta t}{2}\left(g(\mathbf{u}(t), t) + g(\mathbf{u}(t + \Delta t), t + \Delta t)\right).$$

Implicit methods are particularly suitable for stiff systems, where explicit methods require prohibitively small time steps.

For implicit schemes, if the function g is linear (or can be linearized), the resulting equation can be written in a matrix form:

$$\mathbf{M}\mathbf{u}(t + \Delta t) = \mathbf{b},$$

where $\mathbf{M}$ depends on the discretization scheme and $\mathbf{b}$ includes contributions from the current time step. One can choose between:

- Direct solvers (e.g., Gaussian elimination, LU decomposition), which yield exact solutions but can be computationally expensive for large systems.
- Iterative solvers (e.g., Jacobi, Gauss-Seidel, or conjugate gradient methods for symmetric systems), which are more efficient for large or sparse systems. Their convergence, however, depends on the properties of $\mathbf{M}$, and they often require preconditioning for optimal performance. For non-symmetric systems, methods such as GMRES (Generalized Minimal Residual) are commonly used.

For nonlinear PDEs, iterative methods like Newton's method (or its variants, such as inexact or quasi-Newton methods) are employed to solve the nonlinear system at each time step. These methods linearize the nonlinear system around the current approximation and iteratively solve the linearized equations.

Example 2.3.3. For the heat equation, a forward (explicit) Euler scheme to approximate the temporal derivative yields:

$$\mathbf{u}(t + \Delta t) = \mathbf{u}(t) + \Delta t \mathbf{A}\mathbf{u}(t) \tag{2.25}$$

while the backward (implicit) Euler scheme gives:

$$\mathbf{M}\mathbf{u}(t + \Delta t) = \mathbf{u}(t) \tag{2.26}$$

with $\mathbf{M} = \mathbf{I} - \Delta t \mathbf{A}$

2.3.4 Key properties of numerical solvers

When evaluating numerical solvers for PDEs, we usually consider three main properties: consistency, stability, and convergence. We provide an intuition of what they entail and refer the reader to Ferziger and Peric (1996) and Strikwerda (1989) for more details.

Consistency measures how well the discrete scheme approximates the continuous PDE. The idea is that if we substitute the exact solution u into the discrete equations, the error introduced (known as the *truncation error*) should tend towards zero as the grid is refined. Mathematically, if the discrete operator is denoted by g_Δ, then the truncation error is given by:

$$\tau_\Delta = \frac{\partial u}{\partial t} - g_\Delta u.$$

The scheme is consistent if:

$$\tau_\Delta = \mathcal{O}(\Delta x^q) + \mathcal{O}(\Delta t^r),$$

where q and r indicate the order of approximation in space and time, respectively. Consistency ensures that, in the limit as $\Delta x, \Delta t \to 0$, the discrete equations converge to the original PDE.

Stability ensures that errors do not grow uncontrollably during the time evolution. For explicit schemes applied to time-dependent problems, stability is often governed by the CFL condition. For example, for an explicit scheme solving the heat equation, stability typically requires:

$$\Delta t \leq \frac{\Delta x^2}{2\alpha}.$$

For linear problems, stability is sometimes analyzed by examining the eigenvalues of the matrix $\mathbf{M}$, ensuring that $|\lambda| \leq 1$. In short, stability guarantees that the numerical solution remains bounded over time.

Convergence is the property that the numerical solution u_Δ approaches the exact solution u as the discretization is refined:

$$\lim_{\Delta x,\, \Delta t \to 0} u_\Delta(x,t) = u(x,t).$$

According to the Lax Equivalence Theorem, for linear problems a numerical scheme that is both consistent and stable will be convergent. This convergence is typically measured in a suitable norm $\|.\|$. For a fully discrete scheme, the error may be expressed as

$$\|u(\cdot,t_n) - u_\Delta(\cdot,t_n)\| \leq C\Big(\Delta x^q + \Delta t^r\Big),$$

where q and r are the orders of accuracy in space and time, respectively.

2.4 Reduction of complexity

With numerical solvers, we can approximately solve a PDE by solving a high dimensional ODE thanks to the spatial discretization. However, the cost of solving this high dimensional ODE can remain prohibitive and we will cover two different solutions to overcome this limitation.

2.4.1 Reduced Order Models

Reduced Order Models (ROMs) are powerful techniques for solving high-dimensional problems by projecting the dynamics onto a lower-dimensional subspace. This reduction significantly decreases computational cost while capturing the essential behavior of the original system. One widely used ROM technique is the Proper Orthogonal Decomposition (POD), which is described below and with more details in Kerschen et al. (2005).

The core idea behind POD is to represent the high-dimensional solution u as a linear combination of spatial basis functions. Initially, the solution is expressed using a set of n fixed basis functions:

$$u(x,t) = \sum_{k=1}^{n} a_k(t)\psi_k(x),$$

where $\psi_k(x)$ are the spatial modes and $a_k(t)$ are the corresponding time-dependent coefficients. The goal is to determine a reduced set of r optimal basis functions, $\{\phi_1, \ldots, \phi_r\}$ with $r \ll n$, so that:

$$u(x,t) \approx \sum_{k=1}^{r} b_k(t)\phi_k(x),$$

with $b_k(t)$ being the new reduced coefficients.

The key challenge is to select $\{\phi_1, \ldots, \phi_r\}$ so as to minimize the reconstruction error while ensuring computational efficiency. To determine these optimal modes, one minimizes the reconstruction error over a collection of snapshots. Specifically, let

$$\mathbf{u}(t_i) = \big(u(x_1, t_i), \ldots, u(x_N, t_i)\big)$$

be the state vector at time t_i, and let

$$\phi_k = \big(\phi_k(x_1), \ldots, \phi_k(x_N)\big)$$

denote the discretized version of the k-th basis function. Then, the optimal basis $\{\phi_1, \ldots, \phi_r\}$ is obtained by solving:

$$\min_{\phi_1,\ldots,\phi_r} \sum_{i=1}^{M} \left\| \mathbf{u}(t_i) - \sum_{k=1}^{r} \langle \mathbf{u}(t_i), \phi_k \rangle \phi_k \right\|^2 .$$

Here, the inner product $\langle \cdot, \cdot \rangle$ is defined appropriately for the discrete setting (e.g., a Euclidean inner product). In practice, these snapshots are organized into a matrix:

$$\mathbf{U} = \begin{bmatrix} \mathbf{u}(t_1) & \mathbf{u}(t_2) & \cdots & \mathbf{u}(t_M) \end{bmatrix} \in \mathbb{R}^{N \times M},$$

where N is the spatial dimension and M is the number of snapshots. The problem then reduces to finding an orthonormal basis $\mathbf{\Phi}_r = [\phi_1, \phi_2, \ldots, \phi_r]$ that minimizes the Frobenius norm of the reconstruction error:

$$\min_{\mathbf{\Phi}_r} \|\mathbf{U} - \mathbf{\Phi}_r \mathbf{\Phi}_r^\top \mathbf{U}\|_F^2.$$

This optimization is efficiently solved by applying the Singular Value Decomposition (SVD, Golub and Reinsch (1970)) to $\mathbf{U}$:

$$\mathbf{U} = \mathbf{W} \mathbf{\Sigma} \mathbf{V}^\top,$$

where $\mathbf{W} \in \mathbb{R}^{N \times N}$ contains the left singular vectors (spatial modes), $\mathbf{\Sigma}$ is a diagonal matrix with singular values $\sigma_1 \geq \sigma_2 \geq \cdots$, and $\mathbf{V} \in \mathbb{R}^{M \times M}$ contains the right singular vectors (temporal modes). The reduced basis is obtained by selecting the first r columns of $\mathbf{W}$, corresponding to the largest singular values.

Once the reduced basis $\mathbf{\Phi}_r$ is obtained, the high-dimensional state can be approximated by projecting onto the reduced space:

$$\tilde{\boldsymbol{u}}(t) = \mathbf{\Phi}_r \mathbf{\Phi}_r^\top \boldsymbol{u}(t),$$

with the reduced coefficients given by $\mathbf{a}(t) = \mathbf{\Phi}_r^\top \boldsymbol{u}(t)$.

For example, consider a linear dynamical system governed by:

$$\frac{d\boldsymbol{u}(t)}{dt} = \mathbf{L}\boldsymbol{u}(t),$$

where $\mathbf{L}$ is a linear operator. By substituting the reduced representation $\boldsymbol{u}(t) \approx \boldsymbol{\Phi}_r \mathbf{a}(t)$ into the equation, one obtains:

$$\boldsymbol{\Phi}_r \frac{d\mathbf{a}(t)}{dt} = \mathbf{L}\boldsymbol{\Phi}_r \mathbf{a}(t).$$

Exploiting the orthonormality condition $\boldsymbol{\Phi}_r^\top \boldsymbol{\Phi}_r = \mathbf{I}_r$, the dynamics reduce to:

$$\frac{d\mathbf{a}(t)}{dt} = \boldsymbol{\Phi}_r^\top \mathbf{L}\boldsymbol{\Phi}_r \mathbf{a}(t).$$

Thus, the original n-dimensional system is reduced to a system of size r, yielding computational savings.

2.4.2 Parameterization

An alternative approach to reducing computational cost involves modifying the governing equations through modeling approximations. This is particularly relevant in turbulence modeling.

Direct Numerical Simulation solves the Navier–Stokes equations without any turbulence modeling, capturing all relevant scales. DNS requires extremely fine spatial and temporal resolution—often dictated by the Kolmogorov scale—and is typically limited to simple geometries due to its high computational cost. Although DNS provides detailed insight into turbulent flows, its computational expense makes it impractical for daily engineering applications, where only essential flow features (such as drag and lift) are of interest (Ferziger and Peric, 1996).

To make the problem tractable, the Navier–Stokes equations are often averaged or filtered. For instance, in the Large Eddy Simulation (LES) approach, the equations are spatially filtered to resolve large-scale dynamics while modeling the effects of smaller scales through a subgrid-scale model. Alternatively, the Reynolds-Averaged Navier–Stokes (RANS) equations are obtained by time-averaging the Navier–Stokes equations, resulting in a set of equations that describe the mean flow. However, the averaging process introduces unclosed terms due to the nonlinear convective terms, necessitating additional modeling assumptions to close the system.

These parameterization techniques strike a balance between fidelity and computational cost, enabling to get numerical simulations that preserve key flow properties. They are particularly useful because they allow us to generate data, reducing the number of full-scale simulations needed to model a phenomenon. Additionally, this

also means that an approximate representation of the phenomenon is already highly valuable for engineers and scientists. This implies that when developing surrogate models with deep learning, while accuracy remains an important criterion, factors such as applicability and speed will also be critical.

Chapter 3

Introduction to Deep Learning

This section adopts a practical view to present Deep Learning. We will first recap briefly the evolution of DL through time Section 3.1. Then we will detail what is a deep neural network Section 3.2 and how to train them in Section 3.3. We will then present the building blocks of Deep Learning in Section 3.4 and finally go over some important learning paradigms in Section 3.5.

3.1 A short history of deep learning

Learning with Deep Neural Networks (DNN) or Deep Learning (DL) (Goodfellow, 2016) is a key subfield of machine learning (Bishop and Nasrabadi, 2006; Hastie et al., 2009) that has driven significant breakthroughs in the ongoing artificial intelligence revolution. Over the past decade, AI systems powered by deep learning have reached or even surpassed human-level performance in various tasks, including video games (Mnih et al., 2013), chess (Silver et al., 2017a), the game of Go (Silver et al., 2017b), scientific challenges such as protein folding (Jumper et al., 2021), and

have recently sparked profound societal and industrial changes with the revolution of Large Language Models (Radford et al., 2019; Achiam et al., 2023; Touvron et al., 2023).

The primary factor behind this success is that deep learning architectures have significantly benefited from advances in computational resources and the availability of large high-quality datasets. Altogether, this enabled to minimize the reliance on handcrafted features or domain-specific knowledge and to learn everything from the data. This embodies the "bitter lesson" articulated by Sutton (2019), which emphasizes that methods capable of leveraging greater computational power through search and learning tend to dominate other techniques in the long run.

The origins of modern deep learning trace back to the late 1950s with the introduction of the Perceptron (Rosenblatt, 1958). This first neural network consisted of a single layer—a linear transformation followed by an activation function—and was inspired by early models of brain activity. This is the first example of feedforward neural networks (FFNs), which perform computations without any loops or cycles. The Perceptron was trained in a *supervised* fashion, using input features $\mathbf{x}$ to predict binary labels $\mathbf{y}$.

In 1982, Hopfield (1982) demonstrated that neural networks could store information in the state of their neurons and retrieve corrupted examples. These networks, minimized an energy function during inference to recover the most probable example. Another milestone for energy-based modeling came with the introduction of Restricted Boltzmann Machines (RBMs) (Ackley et al., 1985), which in contrast with Hopfield networks, are generative models designed to model probability distributions of *observed* inputs $\mathbf{x}$ with *unobserved* latent variables $\mathbf{z}$ in an *unsupervised* manner. Both Hopfield and Hinton were recently awarded a Nobel prize in physics in 2024, as their works were deemed precursors to contemporary self-supervised learning methods and of modern generative AI techniques.

Perhaps the most influential advancement in deep learning during the late 1980s was the popularization of the backpropagation algorithm (Rumelhart et al., 1986) for training multi-layer perceptrons (MLPs). This method provided a principled approach to training neural networks composed of several layers, by efficiently computing gradients for weight updates. Its effectiveness was demonstrated in combination with convolutional neural networks (CNNs) for handwritten digit recognition (LeCun et al., 1989). These advances marked a significant victory for the field given the computational limitations of the time.

However, in the 1990s and 2000s deep learning struggled to compete with machine learning algorithms such as Support Vector Machines (SVMs) (Cortes and Vapnik, 1995). Challenges such as exploding/vanishing gradients, overfitting, and limited training data hindered its development.

Eventually, the exponential growth of available computational resources and the explosion of web data led to the paradigm shift we have witnessed. A decisive victory

for deep learning occurred in 2012, when the CNN AlexNet (Krizhevsky et al., 2012) won the ImageNet classification competition (Deng et al., 2009) outperforming all the other machine learning techniques.

3.2 Deep Neural Networks

Let us now provide a definition of the general structure of a Deep Neural Network. To keep it simple, we will use the multi-layer perceptron (MLP) as a canonical example of a feed-forward neural network.

Definition 3.2.1. (Multi-Layer Perceptron) For an MLP with L layers and an input $\mathbf{a}^{(0)} = \mathbf{x} \in \mathbb{R}^{d_0}$, the *forward pass* computes the output iteratively for each layer $l = 1, 2, \ldots, L$:

$$\mathbf{h}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}, \quad \mathbf{a}^{(l)} = \sigma^{(l)}(\mathbf{h}^{(l)}), \tag{3.1}$$

where $\mathbf{W}^{(l)} \in \mathbb{R}^{d_l \times d_{l-1}}$ denotes the weight matrix of layer l, $\mathbf{b}^{(l)} \in \mathbb{R}^{d_l}$ is the bias vector of layer l, and $\sigma^{(l)}(\cdot)$ represents the activation function of layer l. The vector $\mathbf{h}^{(l)} \in \mathbb{R}^{d_l}$ is referred to as the pre-activation, while $\mathbf{a}^{(l)} \in \mathbb{R}^{d_l}$ is the post-activation or activation of the output of layer l. The output of the network is therefore $\mathbf{a}^{(L)}$. We also call $\mathbf{h}^{(l)}$ the hidden representations or features of the network.

The activation functions are non-linear operations applied element-wise to the components or *channels* of a vector, and in a standard MLP, the same activation function is typically used throughout the network, such that $\sigma_1 = \sigma_2 = \ldots = \sigma_{L-1}$.

The choice of activation function plays a critical role in training an MLP and continues to be an active area of research. If these activations were replaced by the identity function, the network would reduce to a simple linear transformation, lacking the expressive power to model complex relationships. Conversely, the right activation function can enhance stability during training and improve performance on specific tasks. The most commonly used activation functions are summarized in Table 3.1.

The Sigmoid function maps inputs to the range $[0, 1]$ and was widely used in early neural networks. However, it suffers from vanishing gradients, which makes training deep networks difficult. The Tanh function is closely related but maps the input to the range $[-1, 1]$, offering outputs centered around zero that mitigate some gradient issues compared to Sigmoid. It is a smooth function with non-zero derivatives of any order, which makes it ideal to represent smooth physical phenomenon. On the other hand the ReLU function (Nair and Hinton, 2010), is a non-smooth function but became the standard in deep learning due to its simplicity and efficiency. However, ReLU can suffer from "dead neurons" (when weights stop updating). To address

this, variants like Leaky ReLU (Maas, 2013) and ELU (Clevert, 2015) were proposed, which allow small gradients for negative inputs.

More recent activations aim to balance smoothness and trainability. The Swish function (Ramachandran et al., 2017), and GELU function (Gaussian Error Linear Unit; Hendrycks and Gimpel (2016)) have gained popularity, particularly in transformers, due to their smooth and differentiable nature. Notably the swish function was selected after an exhaustive empirical search over a large set of candidate activation functions. Finally, advanced gating mechanisms like GLU (Dauphin et al., 2017) and SwiGLU (Shazeer, 2020) have been widely adopted in language modeling architectures.

Table 3.1: Common activation functions and their formulas.

Activation name	**Function**
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$
Tanh	$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
ReLU	$\mathrm{ReLU}(x) = \max(0, x)$
Leaky ReLU	$\mathrm{LeakyReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases}$
Softplus	$\mathrm{Softplus}(x) = \ln(1 + e^{x})$
ELU	$\mathrm{ELU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^{x} - 1) & \text{if } x \leq 0 \end{cases}$
Swish (SiLU)	$\mathrm{Swish}(x) = x \cdot \sigma(x), \quad \sigma(x) = \frac{1}{1+e^{-x}}$
Softmax	$\mathrm{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$
GELU	$\mathrm{GELU}(x) = x \cdot \Phi(x), \quad \Phi(x) = \frac{1}{2}\left(1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$
GLU	$\mathrm{GLU}(\mathbf{x}) = \mathbf{x}_1 \cdot \sigma(\mathbf{x}_2), \quad \mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2]$
SwiGLU	$\mathrm{SwiGLU}(\mathbf{x}) = \mathrm{Swish}(\mathbf{x}_2) \cdot \mathbf{x}_1, \quad \mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2]$

3.3 Training Deep Neural Networks

In Definition 3.2.1, we describe the class of models representing MLPs of a specific *depth* L and *width* d_1, but we do not specify which weights $\mathbf{W}^{(l)}$ and biases $\mathbf{b}^{(l)}$ to use. These weights and biases determine the specific behavior of the network and

must be tailored for the task at hand. In the context of statistical learning, the network's knowledge or aptitude to perform a given task is encoded in its weights and is evaluated with a *loss* function and *metrics* relevant to the task. This refers to the paradigm of *empirical risk minimization* (ERM).

Definition 3.3.1. Given a supervised training dataset $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$ consisting of n input-output pairs, where $\mathbf{x}_i \in \mathbb{R}^{d_{\text{in}}}$ is the input vector, $\mathbf{y}_i \in \mathbb{R}^{d_{\text{out}}}$ is the corresponding target, ERM seeks to find the weights θ of a neural network $f_\theta : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$ that minimize the *empirical risk*, defined as the average loss over the training data:

$$\mathcal{R}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(f_\theta(\mathbf{x}_i), \mathbf{y}_i), \tag{3.2}$$

where: $\mathcal{L} : \mathbb{R}^{d_{\text{out}}} \times \mathbb{R}^{d_{\text{out}}} \to \mathbb{R}^+$ is the loss function, such as the mean squared error (MSE) for regression or cross-entropy loss for classification. However, the true objective is to minimize the *expected risk*:

$$r(\theta) = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim p_{\text{data}}}[\mathcal{L}(f_\theta(\mathbf{x}), \mathbf{y})], \tag{3.3}$$

where p_{data} is the true (but unknown) data distribution. Since p_{data} is unknown, the empirical risk $\mathcal{R}(\theta)$ serves as an approximation of the expected risk $r(\theta)$, assuming the training dataset is representative of the true distribution.

From this definition, we can introduce fundamental concepts for deep learning. The primary objective is to train a neural network with low expected risk, while achieving low empirical risk is secondary. This distinction is crucial because neural networks are often overparameterized, increasing the risk of the model simply *memorizing* the dataset instead of *learning* underlying patterns. Our ultimate goal is to perform well in terms of expected risk rather than empirical risk.

This implies that evaluating a neural network's performance solely on the training set, which is used to optimize the weights, is insufficient. Instead, we should use *test* samples $(\mathbf{x}, \mathbf{y}) \sim p_{\text{data}}$ not seen before to estimate the expected risk. This challenge is common across all machine learning applications: when a model has low empirical risk but high expected risk, it has *overfitted* the dataset. Conversely, high empirical risk indicates *underfitting*.

To address overfitting, we can add penalty terms or constraints to the empirical risk, a technique known as explicit regularization. Explicit regularization includes methods such as L_1 or L_2 regularization (Ng, 2004), which penalize large weights, and dropout (Srivastava et al., 2014), which randomly drops activation units during training to prevent overfitting.

In practice, the process of determining suitable weights for a task—and thereby

minimizing an empirical risk function $\mathcal{R}$—typically begins with *initializing* the weights using a *good guess* and refining them iteratively through gradient-based optimization. This phase is called the *training* of a neural network. A *good guess* refers to a variety of initialization schemes, most of which are designed to ensure numerical stability and preserve the norms of hidden representations throughout the network. This helps prevent issues like vanishing or exploding gradients, as achieved by techniques such as Xavier initialization (Glorot and Bengio, 2010) and Kaiming initialization (He et al., 2015b). Proper initialization is especially critical in deep networks, where instability during training can severely hinder convergence.

The optimization process itself relies on backpropagation (Rumelhart et al., 1986), an algorithm that computes the derivatives of the loss function with respect to each network parameter using the chain rule of calculus. By propagating gradients from the output layer to the input layer, backpropagation enables the network to iteratively adjust its weights to minimize the loss. This method is computationally efficient and particularly advantageous in cases where $d_L < d_0$, as gradients at the output layer involve fewer dimensions, reducing overall computational cost.

Example 3.3.1. To illustrate backpropagation, consider a 2-layer neural network with ReLU activation. The network takes an input $\mathbf{x} \in \mathbb{R}^{d_{\text{in}}}$ and outputs $\hat{\mathbf{y}} \in \mathbb{R}^{d_{\text{out}}}$. The forward pass in matrix form is:

$$\mathbf{h}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}, \quad \mathbf{a}^{(1)} = \text{ReLU}(\mathbf{h}^{(1)}), \quad \hat{\mathbf{y}} = \mathbf{W}^{(2)}\mathbf{a}^{(1)} + \mathbf{b}^{(2)}.$$

We use the mean squared error (MSE) loss:

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = \sum_{k=1}^{d_{\text{out}}} (y_k - \hat{y}_k)^2.$$

The weight and bias are updated with gradient descent to minimize the loss $\mathcal{L}$ with a learning rate η :

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \eta \frac{\partial \mathcal{L}}{\partial W_{ij}^{(l)}}, \quad b_i^{(l)} = b_i^{(l)} - \eta \frac{\partial \mathcal{L}}{\partial b_i^{(l)}}$$

We can compute these gradients analytically using the chain rule:

$$\frac{\partial \mathcal{L}}{\partial \hat{y}_k} = 2(\hat{y}_k - y_k), \quad \frac{\partial \mathcal{L}}{\partial W_{ki}^{(2)}} = \frac{\partial \mathcal{L}}{\partial \hat{y}_k} \cdot a_i^{(1)}, \quad \frac{\partial \mathcal{L}}{\partial b_k^{(2)}} = \frac{\partial \mathcal{L}}{\partial \hat{y}_k}.$$

For the hidden layer:

$$\frac{\partial \mathcal{L}}{\partial a_i^{(1)}} = \sum_{k=1}^{d_{\text{out}}} \frac{\partial \mathcal{L}}{\partial \hat{y}_k} \cdot W_{ki}^{(2)}, \quad \frac{\partial \mathcal{L}}{\partial h_i^{(1)}} = \frac{\partial \mathcal{L}}{\partial a_i^{(1)}} \cdot \mathbf{1}_{h_i^{(1)}>0}.$$

For the input layer:

$$\frac{\partial \mathcal{L}}{\partial W_{ij}^{(1)}} = \frac{\partial \mathcal{L}}{\partial h_i^{(1)}} \cdot x_j, \quad \frac{\partial \mathcal{L}}{\partial b_i^{(1)}} = \frac{\partial \mathcal{L}}{\partial h_i^{(1)}}.$$

We can therefore compute the errors with the loss function and propagate this signal to update the weights with this method.

As demonstrated in Example 3.3.1, defining the forward pass of a neural network implicitly defines the backward pass, which is essential for backpropagation during training. In practice, this means that simply implementing the forward pass is insufficient for training; the computation of derivatives is also required to update parameters via backpropagation. Modern automatic differentiation frameworks such as TensorFlow (Abadi et al., 2016), PyTorch (Paszke et al., 2019a), and JAX (Bradbury et al., 2018) efficiently handle this process by dynamically computing and caching the necessary partial derivatives during the forward pass for use in the backward pass.

Optimizing the empirical risk using gradient descent on the entire dataset at once is impractical due to memory and computational constraints. Instead, we use mini-batches $\mathcal{B}$ of size b to compute weight updates. This approach, known as stochastic gradient descent (SGD, Robbins (1951)), updates the parameters using a limited subset of data that fits in memory:

$$\theta \leftarrow \theta - \frac{\eta}{b} \sum_{i \in \mathcal{B}} \nabla_\theta \mathcal{L}(f_\theta(\mathbf{x}_i), \mathbf{y}_i).$$

SGD has been shown to enhance the generalization capabilities of neural networks, as its noisier updates introduce stochasticity that prevents overfitting (Zhang et al., 2016). However, training with SGD can be unstable and slow in practice. To address this, momentum-based methods (Polyak and Juditsky, 1992) are commonly employed to stabilize and accelerate convergence. Momentum involves maintaining an exponential moving average of past gradients in a buffer, effectively smoothing the updates. The momentum buffer is updated iteratively as:

$$m_1 = \beta_1 m_1 + (1 - \beta_1) \frac{1}{b} \sum_{i \in \mathcal{B}} \nabla_\theta \mathcal{L}(f_\theta(\mathbf{x}_i), \mathbf{y}_i).$$

This approach ensures that the gradient vector's direction is influenced by a larger portion of the dataset, improving stability. Variants like RMSProp extend

this idea by tracking a moving average of the squared gradients, allowing for adaptive learning rates:

$$m_2 = \beta_2 m_2 + (1 - \beta_2)\frac{1}{b}\sum_{i\in\mathcal{B}}(\nabla_\theta \mathcal{L}(f_\theta(\mathbf{x}_i), \mathbf{y}_i))^2.$$

The Adam optimizer (Kingma, 2014) combines both momentum and adaptive learning rate ideas, making it a robust and widely used choice for neural network training. Adam is often a good starting point for experimentation, as it balances simplicity and effectiveness across a wide range of tasks.

Even though recent optimizers include adaptive learning rate mechanisms, it remains important in practice to tune the base learning rate η and define it as a function of the optimization step. The general principle is to ensure $\eta_{\text{step}} \to 0$ as step reaches its final stages during the training, facilitating smoother convergence.

The linear decay scheduler reduces η at a constant rate, while the cosine scheduler has initially a more gradual reduction, followed by a steeper decrease towards the end of training. For training transformers and similar architectures, it is often beneficial to initialize with $\eta_0 = 0$ and linearly increase the learning rate to a peak value $\eta_{\max}$ before starting the decay phase. This initial phase, known as *warmup*, helps stabilize training by allowing the optimizer to aggregate sufficient statistics for more reliable momentum estimation.

3.4 Building blocks of Deep Learning

So far, we have examined the fundamental components of a standard neural network and how to train it using backpropagation, focusing on the case of the multilayer perceptron (MLP). While the MLP serves as a canonical example, it is not ideal for all tasks because it lacks specific *inductive biases*.

To overcome these limitations, neural network architectures are constructed using various modules or blocks, which in turn are composed of smaller units called layers. These layers act as the "Lego blocks" for processing data, enabling the design of networks tailored to specific tasks.

In this section, we will introduce the most essential types of layers, then discuss common building blocks, and finally explore some of the key architectures in modern deep learning.

3.4.1 Layers

Definition 3.4.1. (Linear) The linear layer is the simplest type of layer. It

changes the dimensions from d_{in} to d_{out}. It is expressed mathematically as:

$$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

where $\mathbf{W}$ represents the weights, $\mathbf{x}$ the input, and $\mathbf{b}$ the bias. It is sometimes called fully connected layer as it connects every input neuron to every output neuron.

For spatial features $\mathbf{x} \in \mathbb{R}^{H \times W \times d}$ (e.g. an image), directly applying a linear layer to project an input of size $H \times W \times d$ to an output space of $H \times W \times d_{\text{out}}$ can be computationally expensive. To reduce memory consumption and computational cost while introducing spatial inductive biases, convolutions provide a more efficient and effective alternative.

Definition 3.4.2 (Convolution)**.** A convolutional layer processes input data using spatial filters (or kernels) to extract local patterns. In the 2D case, each filter slides over the input $\mathbf{x} \in \mathbb{R}^{H \times W \times d_{\text{in}}}$, applying a cross-correlation operation to compute the output $\mathbf{y} \in \mathbb{R}^{H \times W \times d_{\text{out}}}$.

The weights of the filters are represented by a tensor $\mathbf{W}_k \in \mathbb{R}^{k \times k \times d_{\text{in}} \times d_{\text{out}}}$, where k is the kernel size, d_{in} is the number of input channels, and d_{out} is the number of output channels. The output of the convolution operation at position (h, w) and output channel i is computed as:

$$\mathbf{y}[h, w, i] = \sum_{c=0}^{d_{\text{in}}-1} \sum_{h'=-\lfloor k/2 \rfloor}^{\lfloor k/2 \rfloor} \sum_{w'=-\lfloor k/2 \rfloor}^{\lfloor k/2 \rfloor} \mathbf{W}_k[h', w', c, i] \cdot \mathbf{x}[h + h', w + w', c] + \mathbf{b}[i],$$

where $\mathbf{b} \in \mathbb{R}^{d_{\text{out}}}$ is a bias term.

At the edges of the image, the cross-correlation operation may become ill-defined due to missing values. To address this, the input is padded with pixels using various strategies. For example:

- *Zero padding* adds pixels with zero values,
- *Periodic padding* wraps the image around using periodic boundary conditions,
- *Reflect padding* mirrors the input to impose zero-gradient boundaries.

The number of pixels added to each side is called the padding size, denoted p. To ensure that the kernel is centered and that the output spatial dimensions match the input, the padding size is typically chosen as $p = \lfloor k/2 \rfloor$.

Convolutions can also be extended to non-regular grids, such as graph-based data. In this case, the neighbors of each node are used to define the local connec-

tivity. Another commonly used layer for processing irregular grids and graph-based data is the message-passing layer, which aggregates and updates node features based on neighborhood information (Bronstein et al., 2021).

Definition 3.4.3. (Message-passing) Let us consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where each node $v \in \mathcal{V}$ has a feature vector $h_v \in \mathbb{R}^d$ and each edge $(u, v) \in \mathcal{E}$ has a feature vector $e_{uv} \in \mathbb{R}^e$. A message passing layer updates node representations by exchanging information between adjacent nodes, leveraging both node and edge features. The message passing layer proceeds in three steps. First, for each edge $(u, v) \in \mathcal{E}$, a message is computed using a learnable function ϕ_{message}, typically an MLP:

$$m_{uv} = \phi_{\text{message}}(h_u, h_v, e_{uv}).$$

Next, each node v aggregates incoming messages from its neighbors $\mathcal{N}(v)$ using a permutation-invariant function such as a sum or mean:

$$M_u = \text{aggregate}(\{m_{uv} \mid v \in \mathcal{N}(u)\}).$$

Finally, the node feature is updated by combining the current state with the aggregated message, using another learnable function ϕ_{update}:

$$h'_u = \phi_{\text{update}}(h_u, M_u).$$

Finally, one of the most influential components in modern architectures is the attention mechanism (Bahdanau, 2014). Attention is a powerful approach for processing sequences, as it does not depend on predefined notions of neighborhoods or meshes, making it highly versatile for various data structures.

Definition 3.4.4. (Attention) An attention layer is an aggregation mechanism with a global receptive field, allowing a sequence of size N to retrieve information from another sequence of size M. Specifically, given queries $\mathbf{q} \in \mathbb{R}^{N \times d_k}$, keys $\mathbf{k} \in \mathbb{R}^{M \times d_k}$, and values $\mathbf{v} \in \mathbb{R}^{M \times d_v}$, the attention layer dynamically focuses on the most relevant parts of the input by assigning weights based on their importance. The attention mechanism is defined as:

$$\text{Attention}(\mathbf{q}, \mathbf{k}, \mathbf{v}) = \text{softmax}\left(\frac{\mathbf{q}\mathbf{k}^T}{\sqrt{d_k}}\right)\mathbf{v},$$

where the softmax ensures that the rows of the resulting matrix sum to 1, producing normalized attention weights.

In practice, attention is commonly used in two forms: self-attention and cross-attention. In self-attention, the queries, keys, and values are derived from the same sequence of features $\mathbf{x} \in \mathbb{R}^{N \times d_x}$, with linear projections applied to

compute $\mathbf{q} = \mathbf{W}_q\mathbf{x}$, $\mathbf{k} = \mathbf{W}_k\mathbf{x}$, and $\mathbf{v} = \mathbf{W}_v\mathbf{x}$. In cross-attention, two sequences are involved: $\mathbf{x} \in \mathbb{R}^{N\times d_x}$ and $\mathbf{y} \in \mathbb{R}^{M\times d_y}$. The queries are derived from $\mathbf{x}$, while the keys and values are derived from $\mathbf{y}$, such that $\mathbf{q} = \mathbf{W}_q\mathbf{x}$, $\mathbf{k} = \mathbf{W}_k\mathbf{y}$, and $\mathbf{v} = \mathbf{W}_v\mathbf{y}$.

To facilitate training and ensure a standardized input range throughout the network, it is beneficial to use normalization layers. These layers adjust the activations to a consistent scale and may include learnable parameters. The first widely adopted normalization technique was batch normalization (Ioffe and Szegedy (2015)), which normalizes activations across a mini-batch of inputs. However, more efficient alternatives have since been proposed, including Layer Normalization (Ba et al. (2016)) and Group Normalization (Wu and He (2018)), which normalize the channels in a single example. More recently, RMSNorm (Zhang and Sennrich (2019)), a computationally cheaper variant of Layer Normalization, has gained popularity in transformer-based architectures.

3.4.2 Blocks

These layers are combined into blocks, which serve as reusable building units for constructing more complex models. By stacking these blocks, neural network architectures can be tailored to address a variety of tasks effectively.

Residual Networks (He et al. (2015a)) introduced the key concept of skip connections, which allow gradients to flow directly through the network during optimization. Its core building block, the Residual Block, adds the input directly to the output of a sequence of layers:

$$\mathbf{y} = \mathbf{x} + \mathcal{F}^{(l)}(\mathbf{x}),$$

where $\mathcal{F}^{(l)}$ represents the transformation applied by a few convolutional layers interleaved with activation functions. This residual connection has been fundamental to the development of deep architectures and has a theoretical connection to the discretization of ordinary differential equations. Omitting the Δt term, the residual block resembles the explicit Euler discretization scheme of an ODE, and this discretized version is consistent under some hypotheses (Marion et al., 2023).

The UNet architecture (Ronneberger et al. (2015b), see Figure 3.1) is another widely recognized example of convolutional neural networks. It adopts an encoder-decoder structure, where the encoder reduces spatial dimensions through downsampling layers, and the decoder restores them with upsampling layers. UNet integrates skip connections between corresponding layers in the encoder and decoder. These connections preserve spatial details while reconstructing the output, making UNet particularly effective for regression tasks such as image segmentation.

Transformers (Vaswani et al. (2017), see Figure 3.2) represent a different class

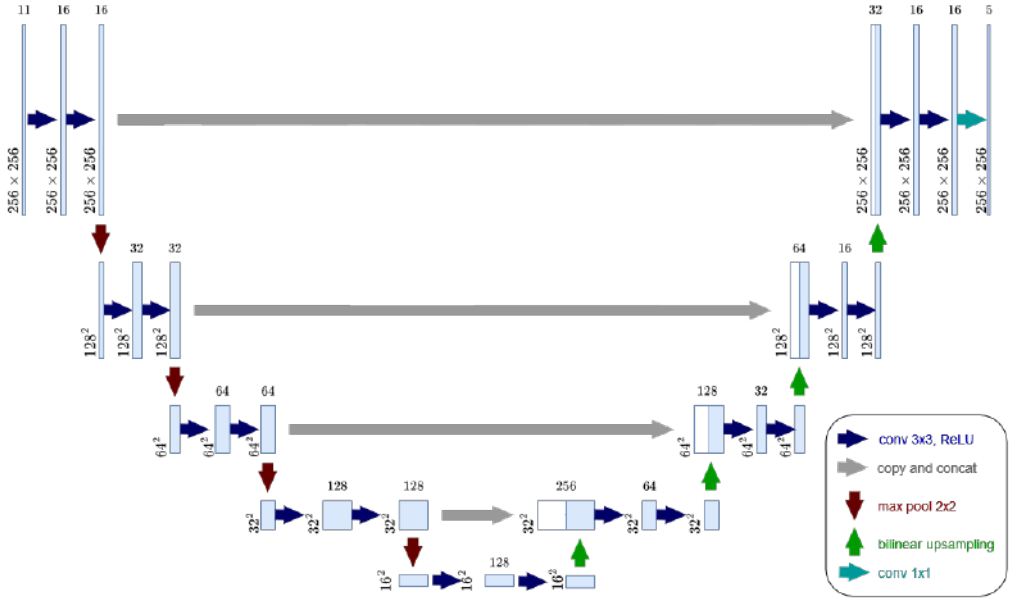


Figure 3.1: UNet architecture. The gray arrows represent skip connections. The red and green represent respectively downsampling and upsampling by a factor 2. Diagram taken from Leung (2020).

of neural networks, designed for processing sequential data. Transformers primarily rely on attention mechanisms, which dynamically focus on different parts of the input sequence. The blocks in Transformers consist of self-attention layers followed by feedforward layers (FFN). This structure enables the model to efficiently capture dependencies in the sequence. Architectures such as GPT (Radford et al. (2018)) adopt this design and achieve state-of-the-art performance in language modeling. GPT employs a transformer block with post-layer normalization and also employs skip connections even within a single block:

$$\begin{aligned} \tilde{\mathbf{y}} &= \text{LayerNorm}(\mathbf{x} + \text{Self-Attention}(\mathbf{x})), \\ \mathbf{y} &= \text{LayerNorm}(\tilde{\mathbf{y}} + \text{FFN}(\tilde{\mathbf{y}})), \end{aligned} \tag{3.4}$$

where $\mathbf{x}$ is the input, $\tilde{\mathbf{y}}$ represents the intermediate output after the attention mechanism, and FFN denotes a two layer feed-forward neural network applied to $\tilde{\mathbf{y}}$.

Recently, pre-layer normalization (Touvron et al., 2023) has proven to be more stable computationally. The difference is that the LayerNorm is applied before the self-attention and feed-forward block.

$$\begin{aligned} \tilde{\mathbf{y}} &= \mathbf{x} + \text{Self-Attention}(\text{LayerNorm}(\mathbf{x})), \\ \mathbf{y} &= \tilde{\mathbf{y}} + \text{FFN}(\text{LayerNorm}(\tilde{\mathbf{y}})), \end{aligned} \tag{3.5}$$

Transformers and attention mechanisms were originally developed for natural

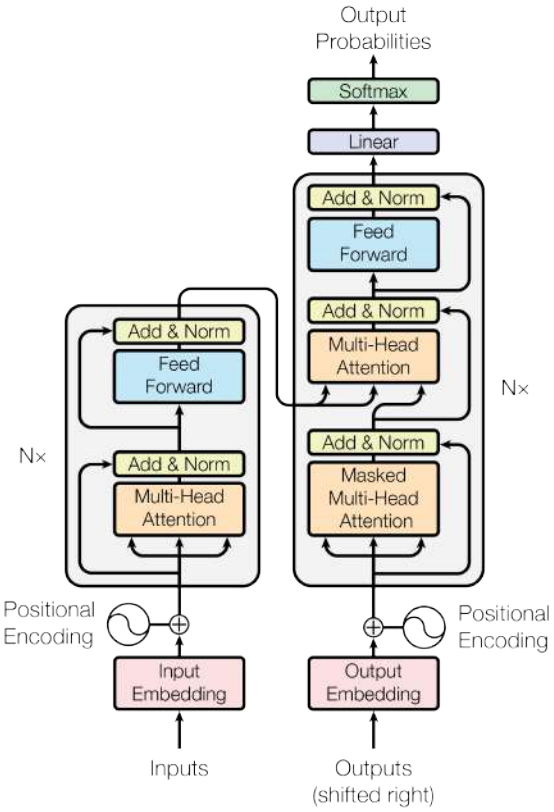


Figure 3.2: Original transformer architecture from Vaswani et al. (2017).

language processing tasks, where they excel at modeling sequences of tokens. This concept was later extended to image processing by Dosovitskiy et al. (2020), who proposed to divide images into small patches of adjacent pixels and treating each patch as a token in a sequence. The Transformer architecture's ability to process and scale efficiently with large datasets has been one of its key strengths.

A major challenge with Transformers is the quadratic complexity of the standard attention mechanism, which scales as $O(N^2)$ with the sequence length N. To address this, several variants have been proposed. For example, Swin Transformers (Liu et al., 2021) or Neighborhood Transformer (Hassani et al., 2023) limit the range of attention to local regions, while other approaches explore alternative attention mechanisms with linear complexity (Katharopoulos et al., 2020). However, the introduction of Flash Attention (Dao et al., 2022) has significantly mitigated these limitations. By optimizing memory usage and minimizing read/write operations, which are the most time-consuming aspects of attention computation, Flash Attention achieves superior efficiency and prevents memory issues. It is however available only on most recent GPUs.

3.5 Learning Paradigms

Before training a deep learning model, we first have to gather and construct a dataset (if it is not already done for us). The choice of learning, i.e. the way we want to train the neural network from data, needs to be aligned with the dataset and the tasks we wish the model to address. So far, we have presented the classical example of supervised learning, where both inputs $\mathbf{x}$ and corresponding targets $\mathbf{y}$ are available. In this setting, the data and the task are well defined: we want to use a model to "predict" the corresponding $\mathbf{y}$ to any new $\mathbf{x}$. However, in many real-world scenarios, such clearly defined inputs and targets are either unavailable or poorly defined. Consider, for instance, random images found on the web: they may lack meaningful annotations beyond a basic caption. Similarly, in the case of a sequence of words, what should the target be for a given word? Should it try to predict another sentence with a question-answer scheme, or should we try to infer the sentiment from this sentence ?

There are therefore many available tasks, but for each task labeled data is often rare as labeling is a time-consuming and expensive process. One of the main strength of deep learning lies in its ability to learn meaningful patterns and representation even when the data might not come under the scope of supervised learning. We can instead adopt different types of learning objectives, and depending on these objectives, models can learn useful feature representations, structural or semantic. They can even learn how to generate new samples.

3.5.1 Unsupervised learning

Unsupervised learning is a classical statistical learning paradigm where we have a dataset of features $\mathbf{X} \in \mathbb{R}^{n \times d}$, where n is the number of samples, and d is the feature dimensionality. However, unlike supervised learning, we do not have explicit labels $\mathbf{Y}$. Several statistical methods can be used in this setting. For example, Principal Component Analysis (PCA, Hotelling (1933)) enables dimensionality reduction by finding a lower-dimensional linear representation of the data that retains the most variance. This is very close to what we presented in Section 2.4.1 to describe the POD in reduced-order modeling. K-Means (MacQueen, 1967) partitions the data into clusters, ideally associating each cluster with meaningful features. Similarly, methods like Gaussian Mixture Models (GMM, Dempster et al. (1977)) attempt to model the underlying data distribution using probabilistic assumptions.

Autoencoder In the deep learning literature, Autoencoders (AE) have become a prominent approach for learning compact representations $\mathbf{z}$ from input features $\mathbf{x}$. Autoencoders generalize the concept of PCA by employing non-linear encoders and decoders to better capture complex data structures. Key milestones in the development of Autoencoders include the original work of Hinton and Salakhutdinov (2006), which demonstrated their ability to learn meaningful low-dimensional representations. Extensions such as Denoising Autoencoders (Vincent et al., 2008) introduced robustness to noise, while Variational Autoencoders (VAE, Kingma and Welling (2013)) incorporated probabilistic modeling to allow the generation of new samples from the latent space. While this latent representation $\mathbf{z}$ can take infinite values, other works have attempted to constrain even further the space of values, with frameworks such as Vector-Quantized Variational Autoencoders (VQVAE, van den Oord et al. (2017)) or VQGAN (Esser et al., 2020), which also includes an adversarial loss developed from the generative adversarial network literature (GAN, Goodfellow et al. (2014)).

Example 3.5.1. (AE, see Figure 3.3) In an autoencoder, the goal is to encode the most informative features of the data $\mathbf{x}$ with an encoder ENC_θ into $\mathbf{z}$ such that the original data $\mathbf{x}$ can be reconstructed with high fidelity with a decoder DEC_ϕ. This yields the following training objective:

$$\mathcal{R}(\theta, \phi) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(\text{DEC}_\phi \circ \text{ENC}_\theta(\mathbf{x}_i), \mathbf{x}_i), \tag{3.6}$$

Although training is typically framed as a dimensionality reduction problem, the learned latent representations $\mathbf{z}_i = \text{ENC}_\theta(\mathbf{x}_i)$ can be leveraged in more complex pipelines or downstream tasks, effectively replacing the original input

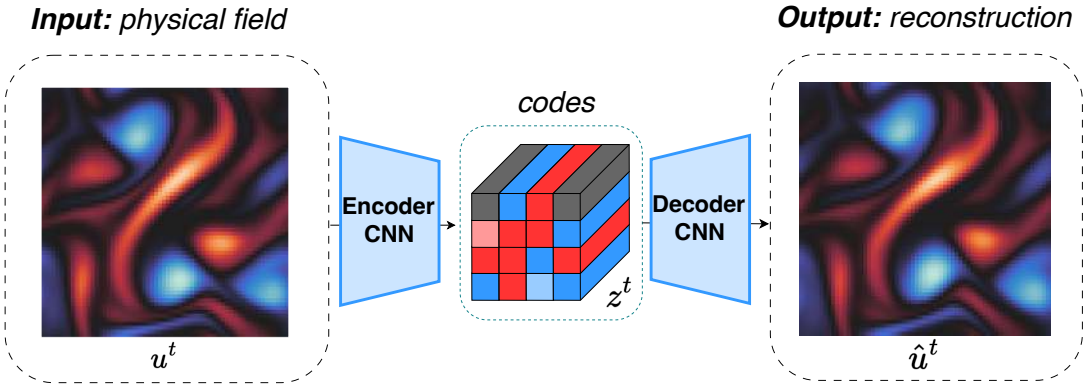


Figure 3.3: Simplified view of an Autoencoder.

features $\mathbf{x}_i$. For example, in the work of Le Naour et al. (2023), it is shown that the indices obtained through the quantization step of a VQVAE can be used to achieve a strong performance in downstream classification tasks.

Neural Fields Implicit Neural Representations (INRs), also known as Neural Fields (NFs), are neural networks designed to learn a continuous representation of a signal, such as an image, a shape, or a video. See Figure 3.4 for an illustration. These coordinate-based networks encode a signal into their weights and decode it by mapping input coordinates to output signal values. This capability makes neural fields particularly powerful for interpolating signals in space or time, enabling applications like super-resolution or the representation of sparse signals.

Despite the apparent simplicity of the task, reconstructing a signal with a continuous neural approach is non-trivial. A straightforward neural network with standard activation functions struggles to represent signals faithfully due to spectral bias (Jacot et al., 2018), which favors learning the low-frequency components. To address this limitation, modern architectures commonly employ frequency-based embeddings (Tancik et al., 2020a; Fathony et al., 2021a; Lindell et al., 2022) or specialized activation functions, such as those used in SIREN (Sitzmann et al., 2020d), to better capture high-frequency details. More recently, some connections have been made between layer normalization in the neural network and the spectral bias of neural fields (Cai et al., 2024).

One of the most prominent applications of neural fields is scene reconstruction, with the famous Neural Radiance Fields (NeRF) (Mildenhall et al., 2021). NeRF is a framework that yields novel views of a scene with smooth and realistic interpolation given a few images of an object captured from different angles. Since its introduction, numerous works have built upon this method. Variable Bitrate Neural Fields (Takikawa et al., 2022a) introduced interpolation schemes leveraging features on regular grids for improved efficiency. Instant Neural Graphics Primitives (Instant

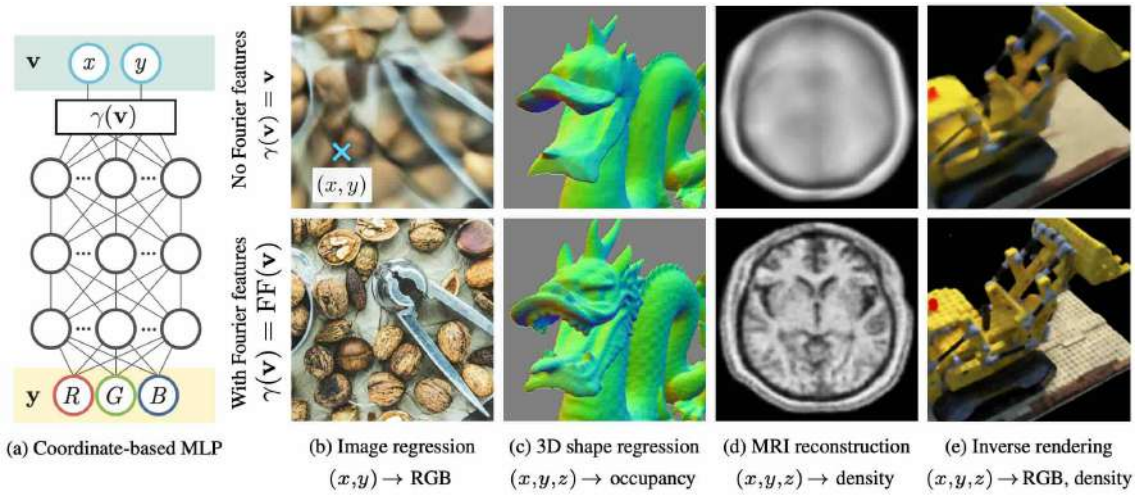


Figure 3.4: Vision applications of Coordinate-based networks or Neural fields (Tancik et al., 2020a).

NGP) (Müller et al., 2022) accelerated training and rendering using a multiresolution hash table, while Shue et al. (2022) reduced the computational cost of scene representation by utilizing triplane features.

More recently, Gaussian Splatting (Kerbl et al., 2023) emerged as a neural-free approach to scene representation. Unlike traditional neural fields, Gaussian Splatting relies on explicit 3D Gaussian rather than neural networks, achieving real-time rendering.

Example 3.5.2. (Neural field) Let us see how we can represent an image $\mathbf{I}$ of size $H \times W$ with a neural field. Each pixel at coordinates (x, y) is normalized betwen $[0, 1]$, with $x = i/W$ and $y = j/H$, where $i \in \{0, \dots, W-1\}$ and $j \in \{0, \dots, H-1\}$. We note the positional embedding $\gamma : \mathbb{R}^2 \to \mathbb{R}^d$ mapping the coordinates (x, y) to a high-dimensional space using a frequency embedding from Mildenhall et al. (2021):

$$\gamma(x, y) = \left[\sin(2^k \pi x), \cos(2^k \pi x), \sin(2^k \pi y), \cos(2^k \pi y)\right]_{k=0}^{L-1}.$$

The neural field $f_\theta : \mathbb{R}^d \to \mathbb{R}^3$ maps $\gamma(x, y)$ to the corresponding RGB values $\mathbf{I}(x, y) = (r(x), g(x), b(x))$:

$$f_\theta(\gamma(x, y)) = (\hat{r}(x), \hat{g}(x), \hat{b}(x)).$$

The training loss is computed with the mean squared error (MSE):

$$\mathcal{R}(\theta) = \frac{1}{HW} \sum_{i=0}^{W-1} \sum_{j=0}^{H-1} \| f_\theta(\gamma(x, y)) - \mathbf{I}(x, y) \|_2^2 .$$

By minimizing $\mathcal{R}(\theta)$, we obtain weights θ such that $f_\theta(\gamma(x, y)) \approx \mathbf{I}(x, y)$, and we can query the network at every coordinate within the spatial domain, even if it is not an observed pixel.

3.5.2 Self-Supervised Learning

Self-supervised learning shares similarities with unsupervised learning but differs in its approach by generating pseudo-labels directly from the data. For instance, given an input $\mathbf{x}$, a label $\tilde{\mathbf{x}}$ can be created by applying transformations or augmentations to $\mathbf{x}$. Another common way involves creating two complementary views of the same input, $\mathbf{x} \to (\tilde{\mathbf{x}}^1, \tilde{\mathbf{x}}^2)$, where for instance $\tilde{\mathbf{x}}^1$ has 90% of the information masked out and $\tilde{\mathbf{x}}^2$ has only the remaining 10%. This approach is particularly useful when reconstructing the input is either too challenging or not the main objective. This focuses rather on building representations that are invariant or consistent across $\tilde{\mathbf{x}}^1$ and $\tilde{\mathbf{x}}^2$.

One of the most impactful aspects of self-supervised learning is its scalability. By leveraging large, unlabeled datasets, self-supervised methods can pretrain models on very large datasets, and can learn complex patterns and structures. This phase, commonly referred to as *pretraining*, is typically the most critical and computationally intensive step in building foundation models such as Llama 3 (Dubey et al., 2024) or Olmo2 (OLMo et al., 2024).

Masking In natural language processing, BERT (Devlin et al., 2018) is perhaps the best example to illustrate the advantages of self-supervised learning. BERT learns rich contextual representations by predicting masked tokens in a sentence of tokens (units for representing words), using an encoder-only transformer architecture. At inference, BERT can encode a sentence to produce contextual embeddings for each token. These embeddings can be used with linear probing to achieve state-of-the-art results in tasks such as sentiment analysis and question answering, outperforming models supervised solely on this task. This framework marked a paradigm shift in deep learning, emphasizing the utility of pretraining on vast datasets in a self-supervised manner. The masking-demasking concept introduced in BERT has since then been extended to audio (Baevski et al., 2020), images (He et al., 2021) and videos (Tong et al., 2022), demonstrating its versatility across domains.

Example 3.5.3. BERT is a foundational model that introduced the concept of masked language modeling (MLM), a paradigm based on masking and demasking parts of a text to train the model on understanding language representations. The training process involves two main objectives: one focuses on guessing missing words in a sequence, while the other predicts whether one sentence follows another. Here, we focus on the masked language modeling aspect.

Let $\mathbf{x} = [x_1, x_2, \ldots, x_N]$ be an input sequence of tokens of length N. A random binary mask $\mathbf{m} = [m_1, m_2, \ldots, m_N]$ is sampled, where each m_j is sampled independently from a Bernoulli distribution with parameter p. A value of $m_j = 1$ indicates that the token x_j will be masked. The resulting corrupted input is denoted $\hat{\mathbf{x}} = [\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N]$, where:

$$\hat{x}_j = \begin{cases} [\text{MASK}] & \text{if } m_j = 1 \\ x_j & \text{otherwise} \end{cases}$$

The MLM objective over a dataset of sequences $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ is to minimize the negative log-likelihood of the original (unmasked) tokens x_{ij} at positions where $m_{ij} = 1$, conditioned on the corrupted input $\hat{\mathbf{x}}_i$. Formally, the training loss is defined as:

$$\mathcal{R}_{\text{MLM}} = -\frac{1}{n} \sum_{i=1}^{n} \mathbb{E}_{\mathbf{m}_i} \left[\sum_{j=1}^{N} m_{ij} \log p_\theta(x_{ij} \mid \hat{\mathbf{x}}_i) \right]$$

Here, $p_\theta(x_{ij} \mid \hat{\mathbf{x}}_i)$ denotes the model's predicted probability of the original token x_{ij}, at position j, given the masked input sequence $\hat{\mathbf{x}}_i$. The expectation is taken over the random masking process, with $\mathbf{m}_{ij}$ sampled independently from Bernoulli distribution for each training step. This objective encourages the model to infer missing tokens from their surrounding context, allowing it to learn rich and robust language representations.

Contrastive Contrastive learning methods, such as SimCLR (Chen et al., 2020b), are also prominent self-supervised learning frameworks in computer vision. These frameworks train models to maximize the similarity between two embedded representations $\mathbf{z}^1$, $\mathbf{z}^2$ from augmented views $\tilde{\mathbf{x}}^1$, $\tilde{\mathbf{x}}^2$ of the same instance while ensuring that representations of different instances remain distinct. For example, SimCLR uses image data augmentations, like random cropping and color distortions, to generate multiple views of an image. Building on this concept, DINO (Caron et al., 2021) integrates a teacher-student framework (Hinton et al., 2015) to stabilize learning and avoiding the need for negative examples. DINO's learned features have been highly effective in tasks such as semantic segmentation and object detection. Beyond computer vision, contrastive learning has been adapted in other modalities such as

time series representations (Franceschi et al., 2019) and even on cross-modal tasks, e.g. aligning images with textual descriptions in CLIP (Radford et al., 2021).

Next-token prediction Another widely used self-supervised objective for training models on sequential data is next-token prediction, where the model learns to predict the next token in a sequence given all preceding tokens as input. By doing so, the model approximates a parameterized distribution over the data by factorizing the joint probability of a sequence into a product of conditional probabilities. This approach forms the cornerstone of pretraining in many large language models (Radford et al. (2018), Radford et al. (2019), Achiam et al. (2023) Touvron et al. (2023), OLMo et al. (2024)).

Architectures trained using the next-token prediction objective offer notable scalability and efficiency. Their scalability stems from employing causal masking in the attention mechanism during training—ensuring predictions rely solely on preceding tokens—and from leveraging dense supervision via parallel processing. Every subsequence in a batch predicts its next token in parallel. Additionally, at inference key-value caching during inference efficiently stores intermediate states, preventing redundant computations.

During inference, these models generate sequences one token at a time based on the provided context. This generative approach is fundamental to the development of robust models capable of handling diverse modalities, including text, images, and videos. For example, it has been extended to tasks such as image generation (ImageGPT Chen et al. (2020a), VQGAN (Esser et al., 2020)) and video generation (MAGVIT2 Yu et al. (2023b), VideoPoet Kondratyuk et al. (2023)).

Most autoregressive generative models work with discrete tokens only, i.e. they model distributions over a finite vocabulary. The *tokenizer* is the block that projects the data onto this finite vocabulary. This is particularly appealing in language modeling, where sentences and words can be efficiently decomposed in subunits. Therefore, it is common to say that next-token predictors model discrete distributions. Nonetheless, recent works such as (Li et al., 2024a) have explored how the next-token prediction objective can be applied to continuous distributions.

Example 3.5.4. Next-token prediction for textual tokens. Consider an input sequence of tokens denoted as $\mathbf{x} = [x_1, x_2, \ldots, x_N]$. The joint probability of the sequence is factorized into a product of conditional probabilities:

$$p_\theta(\mathbf{x}) = \prod_{j=1}^{N} p_\theta(x_j \mid \mathbf{x}_{<j}),$$

where $P(x_t \mid \mathbf{x}_{<t})$ is the probability of the token x_t, conditioned on all preceding tokens. The next-token prediction objective minimizes the negative log-

likelihood of the sequence:

$$\mathcal{R}_{NTP} = -\frac{1}{nN}\sum_{i=1}^{n}\sum_{j=1}^{N}\log p_\theta(x_{ij} \mid \mathbf{x}_{i,<j}),$$

where $p_\theta(x_{ij} \mid \mathbf{x}_{i,<j})$ is estimated using a causal transformer that processes the sequence with attention masking to ensure predictions are only based on preceding tokens.

Diffusion For modalities such as images and videos—where discretization is less straightforward—state-of-the-art generative models are often trained using denoising procedures. During training, inputs are corrupted with noise at varying variance levels, and the model learns to predict the noise injected into the input. Denoising Diffusion Probabilistic Models (DDPM) (Ho et al., 2020) demonstrated that even a simple denoising objective can effectively train models to generate high-quality images starting from pure noise. Building on this, Song et al. (2020) introduced a more efficient sampling procedure, enabling the generation of high-quality images with fewer steps.

Training diffusion models directly in the pixel space, however, is computationally expensive. To address this challenge, Rombach et al. (2021) proposed a latent diffusion framework that stabilizes training by operating in the latent space of autoencoders. This approach not only reduces computational costs but also supports text conditioning, allowing for more flexible generation. Although these models are trained for image generation, they can be adapted for a variety of tasks. For example, diffusion models not only support image editing but also can extract robust features that excel in downstream tasks like image segmentation. Xu et al. (2023) demonstrated that these features can be effectively repurposed to train segmentation models, while DiffCut (Couairon et al., 2024b) shows that diffusion features can achieve state-of-the-art zero-shot segmentation performance.

Recently, flow-matching based methods have emerged as a more sample-efficient alternative for inference. In flow matching, the model learns to approximate the derivative of a flow map that transforms random noise into images (Lipman et al., 2022). Notably, flow matching models solve an ordinary differential equation during inference to generate samples, drawing interesting connections with ODE solvers. Techniques such as Rectified Flows Liu et al. (2022b) have further reduced the number of required sampling steps, and this approach has been successfully scaled, as demonstrated in Esser et al. (2024).

On the architectural side, Peebles and Xie (2023) introduced the Diffusion Transformer (DIT), a simple yet scalable architecture that outperforms traditional CNN diffusion models. This transformer-based architecture has also been adapted for video generation, as showcased by Sora (OpenAI, 2024), highlighting the potential

for scalable generative modeling across diverse modalities.

3.5.3 Meta-learning

Given the diversity of tasks in certain domains, training separate models for each task can be inefficient and resource-intensive. This challenge motivated the development of *multi-task learning*, where a single model is trained to handle multiple tasks simultaneously, sharing representations across tasks to improve generalization. The concept of multi-task learning was introduced in Caruana (1997), where models learn shared parameters across related tasks to improve performance. However, conventional multi-task learning typically involves training on a predefined and fixed set of tasks. When confronted with a new task, the model often requires additional fine-tuning on new task-specific examples, and this stage can be costly and possibly inefficient.

Meta-learning extends multi-task learning by focusing on training a model not to perform specific tasks directly, but to adapt quickly and efficiently to new tasks. In essence, meta-learning is the process of "learning to learn" rather than "learning to do." This paradigm enables models to generalize its learning strategy to unseen tasks, providing greater flexibility and faster adaptation compared to fine-tuning large task-specific models.

In deep learning, one prominent subfield of meta-learning (Hochreiter et al., 2001) is gradient-based meta-learning, where the model is trained to adapt its parameters through gradient updates on a new task. In MAML (Finn et al., 2017), the model learns an initialization of parameters that can be quickly adapted to new tasks with just few gradient steps. There exists many alternatives, such as the first order variant (Reptile, Nichol et al. (2018)) that reduces memory constraints. Meta-sgd (Li et al., 2017a) aims at learning not only the intitialization but also the learning rate for each parameter. Flennerhag et al. (2019) learns to warp gradients to precondition the gradient descent of the network, this can be seen as a generalisation of the previous works. Some lighter alternatives use *context* parameters instead of full parameters to adapt, such as CAVIA (Zintgraf et al., 2019a), where each task has a specific parameter while the main parameters are shared across tasks. .

Auto-decoding Auto-decoding provides an elegant connection between gradient-based meta-learning, multitask learning, and neural fields. It enables the training of autoencoders without the need for an explicit encoder network, which is especially useful when the optimal structure of the encoder is uncertain, but the desired properties of the decoder are well known. For example, this is particularly advantageous for scenarios requiring fully continuous representations of data. By employing a continuous decoder (such as a neural field), auto-decoding derives latent representations directly from the data but without relying on meshes or grids. These representations

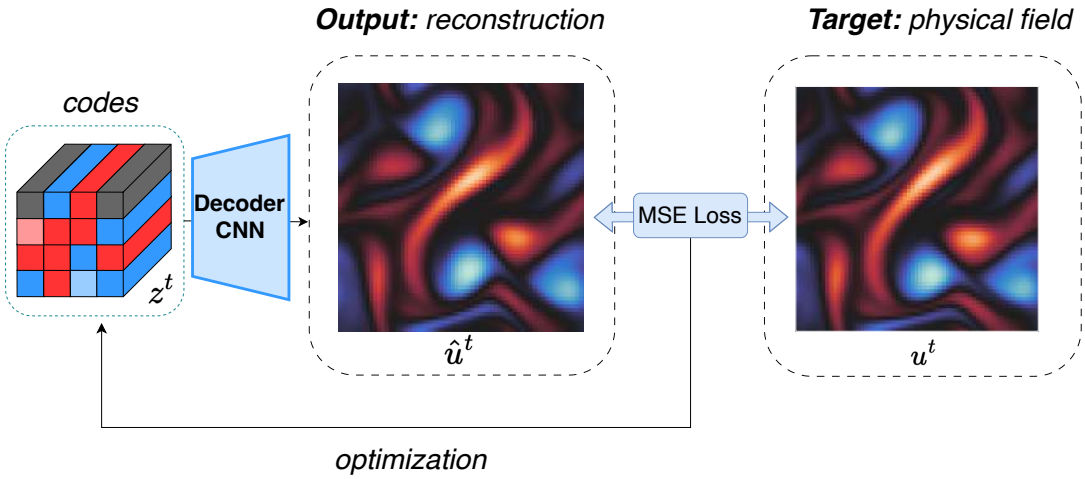


Figure 3.5: Simplified view of an Auto-decoder during inference. The latent codes are optimized to minimize the mean squared error (MSE) loss between the reconstructed output and the input to be encoded.

are encoded within the weights of the neural network and can be adjusted in size by varying the number of context parameters as in Zintgraf et al. (2019b). At inference, auto-decoding replaces the encoder with an optimization loop that optimizes some parameters to obtain the latent representation of the inputs (see Figure 3.5).

The first work by Park et al. (2019b) introduced this concept to generate latent representations of point clouds representing shapes. While this work emphasizes multitask learning rather than meta-learning, it laid the foundation for using neural fields for such scenarios. Subsequently, Sitzmann et al. (2020a) and Tancik et al. (2021), employed meta-learning techniques to enable rapid adaptation across tasks for shapes and images,. These works highlight the potential of combining auto-decoding with meta-learning for efficient and flexible neural representations.

More recent works, including Dupont et al. (2022b) and Dupont et al. (2022a), extend this concept in conjunction with context-based meta-learning by incorporating activation modulations (Perez et al., 2018b) and hypernetworks (Ha et al., 2016). These methods focus on deriving compact latent representations that are both efficient to store and effective for downstream tasks.

While auto-decoding offers a conceptually elegant approach, it can be computationally intensive during both training and inference and may exhibit instability, hindering its scaling potential. Efforts to improve its scalability, such as Bauer et al. (2023), have seen limited success. To address these challenges, several works have explored alternative encoder and decoder designs tailored to neural fields. For instance, Chen and Wang (2022) propose an encoder that transfer patches of data into the weights of an implicit neural network, while Luigi et al. (2023) introduce an encoder that compresses implicit neural representations, which are subsequently

decoded with a conditioned neural field.

3.5.4 In-context learning

The rise of large language models and foundation models has revealed their impressive in-context learning (ICL) capabilities. These models can handle a wide range of tasks, including those not seen during training. ICL refers to a model's ability to solve tasks by leveraging a few examples given in the input prompt. Since the number of examples is limited by the model's maximum sequence length, ICL is closely tied to few-shot learning, as shown by Brown et al. (2020) and further explored by Touvron et al. (2023). Notably, LLMs can even perform well on abstract tasks, such as mathematical reasoning (Wei et al., 2022).

Unlike gradient-based learning, which updates model parameters through training or fine-tuning, ICL adapts by processing examples in the input without modifying its parameters. This makes it a "training-free" approach. Studies such as Liu et al. (2022a) and Mosbach et al. (2023) compare ICL to fine-tuning, finding that while ICL performs comparably for large models, smaller models (fewer than 7 billion parameters) benefit more from fine-tuning.

ICL's effectiveness depends on the pretraining quality and has been linked to meta-learning. Min et al. (2021) introduces a meta-learning approach where models are fine-tuned to use in-context examples effectively. Similarly, Shi et al. (2023) proposes in-context pretraining, where sequences are structured around related tasks rather than random documents during pretraining, improving task contextualization.

Prompt design is crucial for ICL performance. Lu et al. (2021c) highlights the sensitivity of ICL to prompt example order and suggests ways to mitigate this. Min et al. (2022) examines how demonstration examples influence ICL, identifying key success factors. Structured prompts, like chain-of-thought prompting (Wei et al., 2022), enhance ICL by breaking complex tasks into intermediate reasoning steps. Essentially, they bias the model to perform a more thorough and extended analysis of the input before generating its final answer.

The mechanisms behind ICL remain partially understood. One theory views ICL as implicit Bayesian inference, where models deduce task-relevant distributions from the context. Another, proposed by von Oswald et al. (2022), suggests that ICL mimics gradient descent, with attention over the context examples approximating parameter updates. Pan et al. (2023) further differentiates between a model's ability to recognize tasks and learn task-specific patterns, noting that once the task is identified, the labels in the examples become less important.

> **Example 3.5.5.** In-context learning enables a model to infer and perform a task by observing examples of the function it is expected to approximate. For

instance, in a color translation task from English to French, providing examples like "blue: bleu, green: vert, pink: rose," followed by "yellow:" allows the model to deduce that "jaune" corresponds to "yellow," just as "vert" corresponds to "green." Alternatively, the instruction can be explicitly stated in natural language, such as "What is the French word for yellow?".

While natural language instructions work well for text-based tasks, they may not sufficiently define the task in other modalities. Consider an image editing scenario where the goal is to add a specific pair of glasses to the face of a person. Rather than describing the glasses in natural language, it seems far more informative to provide a set of example images showing people with and without the target pair of glasses. In such cases, presenting structured examples of the task provides a more effective and unambiguous way to condition the model than relying on potentially vague textual descriptions.

Chapter 4

Deep Learning for solving PDEs

Now that we have covered the key concepts of partial differential equations and deep learning, this section will explore the existing methods that leverage deep learning to solve PDEs or, more broadly, to model spatiotemporal dynamics.

Broadly speaking, these methods fall into three main categories. The first includes approaches that incorporate physical information into deep learning architectures. The second involves techniques designed to augment or accelerate traditional numerical solvers by integrating components based on deep learning. Finally, and most relevant to our focus, are the methods that adopt a data-centric approach, relying mainly on data rather than explicit physical principles.

4.1 Physical priors for Deep Learning

Hard constraints Bézenac et al. (2017) were the first to integrate physical priors into a deep learning architecture. They focused on an advection-diffusion system to model sea surface temperature (SST), leveraging the formulation of its analytical solution to compute the temperature field at time $t + \Delta t$ as a convolution of the temperature field at time t with a Gaussian kernel dependent on the advection flow $w(t)$. Since this flow is not directly observed, they modeled it using a UNet and trained the neural network to predict future time steps. This work represents an early example of enforcing *hard constraints* in neural networks, here ensuring that

outputs adhere to known analytical solutions. More broadly, hard constraints can be imposed to enforce physical laws or boundary conditions, for example through iterative solvers in Negiar et al. (2022); Chalapathi et al. (2024).

Soft constraints However, designing explicitly a network to satisfy a constraint is difficult in most cases. As this is not trivial we can penalize the network with *soft constraints* instead. This transforms the problem of finding a solution of a PDE into an optimization problem. This is the line of works started with the Deep Galerkin Method (Sirignano and Spiliopoulos, 2018) and the Physics Informed Neural Networks (PINNs) (Raissi et al., 2019), which use a combination of loss terms to enforce the partial differential equation or the boundary condition. Though this approach sounds simple and convenient thanks to automatic differentiation, in practice it is very difficult or even infeasible to obtain acceptable PDE solutions, as the optimization problem is ill-conditioned (Krishnapriyan et al., 2021; De Ryck et al., 2023). A simple workaround, though not always applicable, is to reduce the number of soft constraints by replacing some with hard constraints when possible (Lu et al., 2021b). However, the optimization process remains ill-conditioned. To address this, Boudec et al. (2025) propose a neural solver that efficiently optimizes the PINN losses by guiding the optimization towards the numerical solver's solution within just a few iterations. This results in an orders-of-magnitude speedup over standard PINN optimization.

Example 4.1.1. We consider the heat equation again, this time with neumann boundary conditions:

$$\begin{aligned} \frac{\partial u}{\partial t} &= \alpha \frac{\partial^2 u}{\partial x^2}, && x \in [0, L], \quad t > 0, \\ u(x, 0) &= u^0(x), && x \in [0, L], \\ u(0, t) &= b_1(t), \quad u(L, t) = b_2(t), && t > 0. \end{aligned} \tag{4.1}$$

In the PINNs framework, we approximate the solution u to this problem using a neural network f_θ and leveraging automatic differentiation. The parameters θ are learned by minimizing the following multi-term loss function:

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{PDE}} + \lambda_{\text{IC}} \mathcal{L}_{\text{IC}} + \lambda_{\text{BC}} \mathcal{L}_{\text{BC}}. \tag{4.2}$$

Each term in the loss function enforces a different constraint of the problem. The loss $\mathcal{L}_{\text{PDE}}$ ensures the solution satisfies the heat equation at collocation points.

$$\mathcal{L}_{\text{PDE}} = \sum_{(x_i, t_i)} \left(\frac{\partial f}{\partial t}(x_i, t_i) - \alpha \frac{\partial^2 f}{\partial x^2}(x_i, t_i) \right)^2. \tag{4.3}$$

Here the derivatives can be automatically computed analytically using automatic differentiation. The initial condition loss $\mathcal{L}_{\text{IC}}$ enforces agreement with the given initial state:

$$\mathcal{L}_{\text{IC}} = \sum_{x_i} \left(f(x_i, 0) - u^0(x_i) \right)^2 . \tag{4.4}$$

Finally, the boundary condition loss $\mathcal{L}_{\text{BC}}$ ensures that the solution respects the prescribed boundary values:

$$\mathcal{L}_{\text{BC}} = \sum_{t_i} \left(f(0, t_i) - b_1(t_i) \right)^2 + \sum_{t_i} \left(f(L, t_i) - b_2(t_i) \right)^2 . \tag{4.5}$$

Symmetries Symmetry in a system arises when certain transformations (e.g., translations, rotations, reflections) leave its fundamental properties unchanged. In the context of partial differential equations, symmetries can be leveraged by considering the solution space itself. Specifically, if u is a solution to an initial-boundary value problem with initial conditions u^0, and there exists a transformation ϕ such that $\phi(u)$ is also a solution with initial conditions $\phi(u^0)$, then this property can be exploited to improve neural surrogate models. This leads to the development of equivariant models, where the conservation of symmetry can be expressed as:

$$g(\phi(u)) = \phi(g(u))$$

This principle has been explored in various ways in the literature. For instance, Wang et al. (2020b) introduced equivariant convolutional layers to enforce PDE symmetries in neural networks for the Navier-Stokes and Heat equations. Brandstetter et al. (2022b) proposed a data augmentation strategy based on symmetry transformations of PDEs, leveraging the fact that if u is a solution, then $\phi(u)$ remains a valid solution for any transformation ϕ in the symmetry group. Similarly, Mialon et al. (2023) investigated the use of PDE symmetries for both data augmentation and self-supervised learning of solution representations, following contrastive learning approaches such as SimCLR (Chen et al., 2020b). However, a key challenge in these methods lies in determining which symmetries to exploit and how similar two transformed views should be in order to learn meaningful representations.

4.2 Hybrid modeling

Incorporating physical knowledge directly into neural networks is a challenging task. As a result, other approaches have instead taken the opposite direction, integrating neural networks into classical solvers. These methods aim to achieve various objectives, such as augmenting partial knowledge of a system (Yin et al., 2021), combining

numerical solvers and neural networks (de Avila Belbute-Peres et al., 2020), or learning more efficient parameterizations to improve closure modeling (Kochkov et al., 2021). Recently, this paradigm has showcased impressive capabilities for complex applications in weather modeling (Kochkov et al., 2023).

This integration typically requires solvers that are compatible with automatic differentiation. There are two main categories of solvers that enable this combination: neural ODE-based solvers and implicit differentiation-based solvers. These can be used with the help of specific libraries such as torchdiffeq (Chen, 2018) or diffrax (Kidger, 2021).

Neural ODEs So far, many architectures designed for scientific machine learning have been inspired by existing models for sequential data or images, such as CNNs and Transformer-based architectures. These models follow a straightforward structure: given an input $\mathbf{x}$, the network applies a series of transformations to produce an output. However, another class of architectures actually draws inspiration from ODEs and PDEs, namely the neural ODE architecture (Chen et al., 2018). It differs from standard feedforward networks by modeling the derivative of a state variable $\mathbf{x}$ with a neural network g_θ:

$$\frac{d\mathbf{x}}{dt} = g_\theta(\mathbf{x}, t) \tag{4.6}$$

where x is the state variable, and t represents time. Namely it is formulated as in Equation (2.13), but g_θ is now a neural network.

This modeling is actually close to the ResNet architecture and skip connections as explained in Section 3.4.2, where ResNet blocks can be compared to the integration steps of a discretized Euler scheme of Equation (4.6). During training and at inference, the state at arbitrary time t_1 can be queried by solving the ODE forward in time : $\mathbf{x}(t_1) = \text{ODESolve}(\mathbf{x}(t_0), t_1 - t_0, g_\theta)$.

There are two training methods for neural ODE models: (i) *Discretize-then-Differentiate* (standard backpropagation). The model is trained like a typical neural network using automatic differentiation. While straightforward, this method requires storing all intermediate states, leading to high memory consumption. (ii) *Differentiate-then-Discretize* (Adjoint Method): Instead of storing intermediate states, the gradients are computed by solving a second ODE backward in time. This approach reduces memory requirements but is computationally more expensive. Different numerical solvers can be employed for these methods, such as Runge-Kutta methods (RK4, Dopri5) and even implicit solvers for stiff systems. In practice, (i) is often preferred.

Implicit Solvers and Differentiation A related approach relies on the implicit function theorem, which enables differentiation through iterative solve operations in

order to compute the solution to a linear system of the form:

$$\mathbf{A}\mathbf{x} = \mathbf{y}$$

This type of system appears frequently in finite element methods (FEM) and other numerical PDE solvers. In this context, neural networks can be combined with the iterative solver while maintaining gradient flow through the steps of the solvers as in Um et al. (2020b) or Negiar et al. (2022).

4.3 Data-centric approaches

Strictly speaking, hybrid modeling also relies on numerical simulations to train neural networks. However, since this section serves as a key foundation before presenting our contributions, we provide a thorough description of the setup here.

Setting Unlike physics-informed methods or numerical solvers that incorporate prior knowledge of the system, these purely data-driven approaches can be used to train models that emulate any kind of phenomena, whether governed by known or unknown equations. This flexibility makes them particularly valuable in real-world scenarios, where large datasets are available, but the underlying physical laws are either partially known or too complex to be fully captured by a set of equations.

In the context of solving PDEs, we will consider this setting, adapted from Definition 2.2.4:

$$\begin{aligned} \frac{\partial u}{\partial t} &= g\big(\mu, x, t, f(x,t), u, \nabla u, \nabla^2 u, \dots\big), \\ &\qquad \forall x \in \Omega, \forall t \in (0,T] \\ \mathcal{B}(u)(x,t) &= 0, \quad \forall x \in \partial\Omega, \forall t \in (0,T] \\ u(0,x) &= u^0, \quad \forall x \in \Omega \end{aligned} \tag{4.7}$$

where g is a function of the solution u and its spatial derivatives on the domain Ω, and also includes the forcing term f ; $\mathcal{B}$ is the boundary condition constraint (e.g., spatial periodicity, Dirichlet, or Neumann) that must be satisfied at the boundary of the domain $\partial\Omega$; and u^0 is the initial condition sampled with a probability measure $u^0 \sim p^0(.)$.

4.3.1 Neural surrogates

A surrogate model is a function g_θ designed to approximate the temporal evolution of a system. In the autoregressive case, this amounts to design a model such that:

$$g_\theta(u^t) \approx u^{t+\Delta t}$$

To do so, the classical training setup for a neural surrogate assumes access to a set of trajectories $u_i^t, u_i^{t+\Delta t}, \ldots, u_i^{t+T}$ where the objective is to train a model that advances the system forward in time. The standard training loss minimizes the empirical risk with a relative loss:

$$\mathcal{R}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}_t \frac{\left\| g_\theta(u_i^t) - u_i^{t+\Delta t} \right\|}{\left\| u_i^{t+\Delta t} \right\|} \tag{4.8}$$

This formulation ensures that the neural surrogate learns to approximate the time-stepping operator. As mentioned before, the main motivation for using a surrogate model is to accelerate the resolution of dynamical systems by replacing computationally expensive numerical solvers with a cheaper alternative. Instead of solving the full system of equations at each time step, one can efficiently predict future states using g_θ, significantly reducing computational cost. In order to recover a trajectory with several snapshots, we can make recursive calls to the solvers:

$$u^{t+k\Delta t} = \underbrace{g_\theta \circ g_\theta \circ \cdots \circ g_\theta}_{k \text{ times}}(u^t).$$

However, as with classical numerical solvers, these neural surrogates can diverge over time. Therefore, the classical evaluation metric is the relative L^2 error over the trajectory:

$$\mathcal{L}^2_{\text{test}} = \frac{1}{n_{\text{test}}} \sum_{j=1}^{n_{\text{test}}} \frac{\left\| \hat{u}_j^{\text{trajectory}} - u_j^{\text{trajectory}} \right\|}{\left\| u_j^{\text{trajectory}} \right\|} \tag{4.9}$$

One major advantage of neural networks over classical solvers is their ability to learn mappings for arbitrary temporal resolutions Δt, enabling much faster inference compared to traditional numerical methods. On the other hand, neural surrogates do not have guarantees at inference and do not perform well for out-of-distribution dynamics.

The two main neural architectures used for training surrogate models on discrete grids are either based on CNN or GNN.

CNN-Based Surrogate Models Convolutions operate locally on fixed-size neighborhoods, drawing similarities to classical numerical schemes used to approximate spatial derivatives in PDEs, such as finite differences. The most commonly used architectures in this context are direct adaptations of classical vision models, notably UNet and ResNet (Gupta and Brandstetter, 2022). To enhance expressivity, Stachenfeld et al. (2022) proposed the use of dilated convolutions, while Zhang et al. (2024) introduced a strategy of stacking multiple UNet blocks to enable a more progressive processing of the input. CNN-based surrogate models are inher-

ently restricted to regular grids due to their reliance on traditional convolutions. To accommodate irregular grids and arbitrary geometries, a common approach is to interpolate observations onto a regular grid or apply masking. However, these processing steps distort the original signal and have been shown to be less effective in the presence of obstacles within the domain, as demonstrated in Li et al. (2022a).

GNN-Based Surrogate Models A more natural approach for handling irregular grids is to represent the mesh as a graph, where observations correspond to graph nodes and edges—defined based on a notion of distance—capture connectivity. In Pfaff et al. (2021), a graph neural network (GNN) with multiple message-passing blocks demonstrated strong performance on both Eulerian and Lagrangian systems. Building on this framework, Brandstetter et al. (2022d) introduced several key enhancements, including the pushforward and temporal bundling tricks, as well as the use of relative distances in message-passing blocks. Moreover, they showed that numerical solver schemes, such as finite differences, can be formulated naturally within a message-passing framework.

Despite their advantages, these surrogate models remain inherently mesh-dependent, making them challenging to apply in certain contexts. CNN-based models are constrained to regular grids, limiting their applicability in real-world industrial problems involving complex geometries. GNN-based models, on the other hand, rely on graph connectivity defined by mesh structures, preventing them from generalizing to changes in topology or discretization levels (Yin et al., 2023; Li et al., 2023b). Training a GNN without downsampling the original mesh is often impractical, mainly due to the high memory complexity inherent to message passing layers on large graphs (Catalani et al., 2024). Furthermore, GNNs also suffer from over-smoothing (Rusch et al., 2023), where node embeddings become indistinguishable after multiple message-passing layers, limiting the capture of fine-scale local features.

These limitations underscore the need for more flexible surrogate models that can generalize across different spatial discretizations while remaining computationally efficient. This motivates the development of neural operators, which we discuss next.

4.3.2 Operator learning

Operator learning generalizes the concept of supervised learning for finite-dimensional vectors to functional representations.

Definition 4.3.1. (Operator Learning) Let $\Omega \subset \mathbb{R}^d$ be a bounded, open set, and define the input function space $\mathcal{A} = \mathcal{A}(\Omega; \mathbb{R}^{d_a})$ and output function space $\mathcal{U} = \mathcal{U}(\Omega; \mathbb{R}^{d_u})$ as separable Banach spaces of functions taking values in $\mathbb{R}^{d_a}$ and $\mathbb{R}^{d_u}$, respectively.

Furthermore, let $G^\dagger : \mathcal{A} \to \mathcal{U}$ be an operator that maps a PDE parameter $a \in \mathcal{A}$ to the corresponding PDE solution $u \in \mathcal{U}$. Suppose we have a training set of observation pairs $\{a_i, u_i\}_{i=1}^n$, the goal of operator learning is to construct a parametric approximation of $G^\dagger$ by learning a map $G_\theta : \mathcal{A} \to \mathcal{U}$ such that $G_\theta \approx G^\dagger$. The optimal parameters θ are found by minimizing the empirical risk:

$$\mathcal{R}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(G_\theta(a_i), u_i), \tag{4.10}$$

where $\mathcal{L}$ is a suitable loss function measuring the error between the predicted and true solutions.

The operator learning setting encompasses various tasks. For instance, by setting $a = u^t$ and defining $G^\dagger(a) = u^{t+\Delta t}$, we recover an autoregressive framework as seen in Section 4.3.1, except that we operate on functions rather than their discretized representations. Alternatively, in non-autoregressive settings, the objective may be to predict the full solution trajectory over a given time horizon T, where $a = u^0$ and $u = u^{(0,T]}$, as in Lu et al. (2021a).

Although operator learning is formulated with a functional perspective, in practice, these functions are discretized over spatial grids $\mathcal{X}_i \subset \Omega$. Consequently, operator learning methods must be robust to variations in the discretization level, ensuring they depend only on the underlying function. Furthermore, they should accommodate arbitrary observation points within the spatial domain Ω.

To address these challenges, several architectures have been proposed for operator learning. DeepONet (Lu et al. (2021a), see Figure 4.1 for an illustration) was introduced to enable continuous querying of the solution within the domain Ω and builds on the separation of variable principles, briefly presented in Section 2.1. The Fourier Neural Operator (FNO, Li et al. (2021), see Figure 4.2 for an illustration) leverages spectral convolutional blocks to capture global spatial interactions through an integral kernel term. The Graph Neural Operator (GNO) (Li et al., 2020b) adopts the same idea by modeling the kernel integral using graph convolutions. More recently, the Convolutional Neural Operator (CNO) (Raoni'c et al., 2023) is a modified UNet architecture, incorporating the resolution-aware activation function from Karras et al. (2021) to improve adaptability across different resolutions. GINO (Li et al., 2023b) combines elements of GNO and FNO to handle larger meshes effectively. Additionally, several operator learning models based on transformers have been introduced (Li et al., 2023a; Hao et al., 2023), employing linear self-attention and cross-attention mechanisms to allow for continuous querying of the solution and flexible processing.

The desired property of discretization invariance for operator learning has been studied in Bartolucci et al. (2024), where they introduce a mathematical framework to analyze the properties of neural operators. They demonstrate that both CNO and SNO (Fanaskov and Oseledets, 2023) have desirable properties when increasing

the resolution beyond that used during the training.

We now describe in more details the two most popular architectures for operator learning.

Example 4.3.1. The DeepONet architecture (Lu et al., 2021b) consists of two neural networks:

- *Branch Network*: Given an input function u observed at discrete points $\{x_1, x_2, \ldots, x_m\}$, the branch network B_θ encodes these values into a feature representation:

$$b(u) = B_\theta(u(x_1), u(x_2), \ldots, u(x_m)) \in \mathbb{R}^p,$$

  where p is the latent dimension.

- *Trunk Network*: Given a query location $y \in \Omega$, the trunk network T_ϕ maps it to a latent representation:

$$t(y) = T_\phi(y) \in \mathbb{R}^p.$$

The final prediction of DeepONet is obtained via a weighted sum of the latent representations, akin to the separation of variables principle:

$$G_\theta(u)(y) = \sum_{i=1}^{p} b_i(u) t_i(y) = b(u)^\top t(y).$$

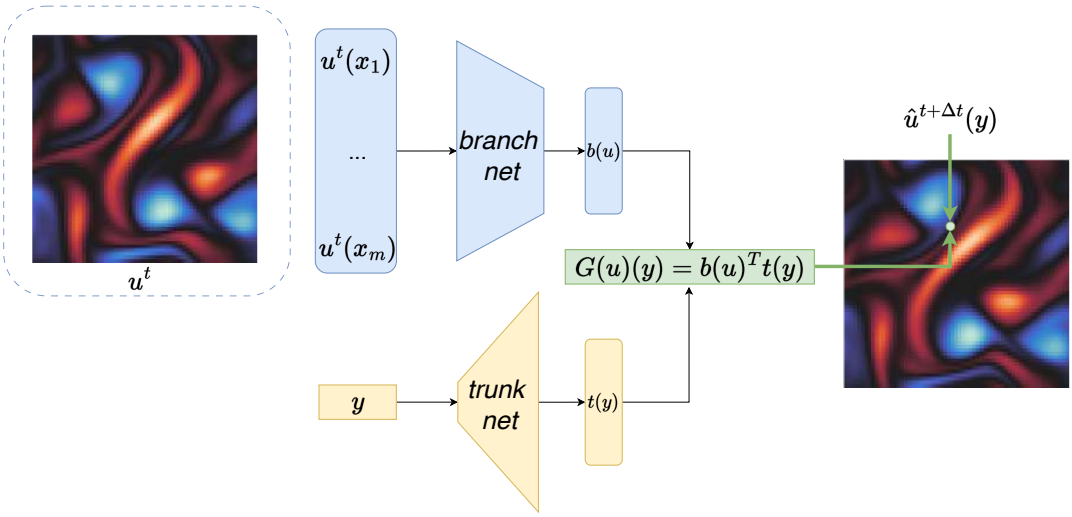


Figure 4.1: Overview of the DeepONet architecture.

Example 4.3.2. The Fourier Neural Operator (FNO) (Li et al., 2020a) is an architecture that performs global convolutions in the frequency domain, enabling efficient long-range interactions. It relies heavily on the Fast Fourier Transform (FFT) and inverse Fast Fourier Transform (IFFT) to go back and forth between the spatial and spectral domain. The architecture consists of three main steps:

- The input function u is first lifted into a higher-dimensional feature space through a pointwise linear transformation:

$$v_0(x_j) = \mathbf{W}^{(0)} u(x_j) \in \mathbb{R}^{d_h}, \quad \forall j = 1, \dots, m,$$

  where d_h is the dimensionality of the feature space.

- A sequence of spectral convolution blocks is applied to the lifted representation. At each block l, the function is transformed into the Fourier domain:

$$\hat{v}^{(l)}(k) = \mathrm{FFT}(v^{(l)}(x_1), \dots, v^{(l)}(x_m)),$$

  where $\hat{v}^{(l)}(k) \in \mathbb{R}^{d_h}$ represents the frequency components at frequency k. Instead of processing all frequency modes, FNO truncates the high-frequency components, retaining only the lowest $k_{\max}$ modes and then applies a linear layer:

$$\tilde{v}_c^{(l)}(k) = \sum_{c'=1}^{d_h} R_{k,c,c'}^{(l)} \hat{v}_{c'}^{(l)}(k), \quad \forall k \in \{1, \dots, k_{\max}\}, \quad \forall c \in \{1, \dots, d_h\}.$$

  This truncation reduces computational cost and prevents overfitting to high-frequency noise, improving generalization. The filtered representation is then transformed back into the spatial domain via the inverse Fourier transform:

$$v^{(l),\dagger}(x_1), \dots, v^{(l),\dagger}(x_m) = \mathrm{IFFT}(\tilde{v}^{(l)}(1), \dots, \tilde{v}^{(l)}(m)).$$

  Finally, the result of the global convolution is added to a linear projection of the input features, and the sum is passed through a nonlinear activation. This residual term allows to keep high-frequency components through the different blocks.

$$v^{(l+1)}(x_j) = \sigma(v^{(l),\dagger}(x_j) + \mathbf{W}^{(l)} v^{(l)}(x_j)), \quad \forall j = 1, \dots, m.$$

- After the Fourier blocks, the final feature representation $v^{(L)}(x)$ is projected back to the target function space using a pointwise linear layer:

$$G_\theta(u)(x_j) = \mathbf{W}^{(L+1)} v^{(L)}(x_j), \quad \forall j = 1, \dots, m.$$

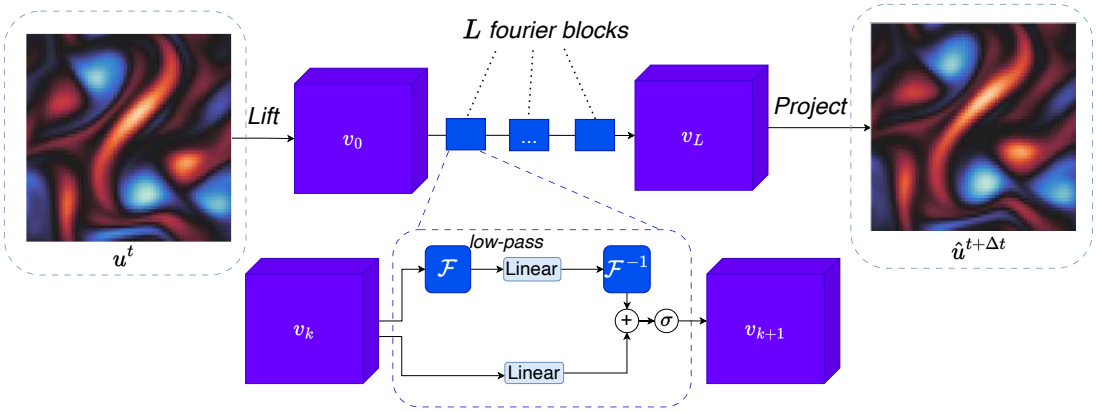


Figure 4.2: Overview of the FNO architecture.

Each architecture has its own strengths and limitations. DeepONet allows the evaluation anywhere in the domain but requires input observations at fixed locations. FNO is efficient but cannot be used with irregular geometries due to its reliance on FFT, the Discrete Fourier Transform being too costly. CNO is robust to changes in the discretization but is limited to regular domains. OFormer and GNOT can process irregular geometries but use simplified attention schemes suited for a processing in the observation space. GNO is flexible but struggles to scale with mesh size, making GINO the preferred choice in practice.

4.3.3 Learning with multiple physical parameters

In previous sections, the main varying parameter of the equation was essentially the initial condition, which means models are evaluated on their ability to unroll a trajectory given a new initial condition u_*^0. Beyond this, evaluations can also include out-of-distribution scenarios, such as new domains with different topologies or geometries, alternative observation samplings that modify resolution or sparsity, and extended time horizons to see temporal extrapolation capabilities.

However, unlike classical solvers that can handle a range of PDE parameters—including scalar coefficients (e.g., viscosity or advection speed), boundary conditions (e.g., Neumann or Dirichlet), and forcing terms—neural networks struggle to generalize to such variations at inference time.

Direct conditioning A straightforward approach to allow for generalization to a larger set of PDE parameters is to condition the neural network directly with those that we expect to vary at inference. This works well when dealing with scalar coefficients or when boundary conditions can be easily enforced, as explored in Brandstetter et al. (2022d) and Takamoto et al. (2023). However, this method assumes precise knowledge of the system's parameters and limits the flexibility of how they condition the network.

Adaptation methods A more principled approach (Yin et al., 2022a; Kirchmeyer et al., 2022; Koupaï et al., 2024) partitions the parameter space into *environments* e_j, where each environment e_j consists of trajectories governed by the same underlying dynamics (i.e., identical PDE parameters except for the initial condition). These methods typically use shared parameters θ across environments while introducing environment-specific parameters ξ_j to capture distinct dynamics. During training the common parameters θ and environment parameters ξ_j are jointly learned. At inference, given an initial condition u_*^0, these methods rely on gradient-based adaptation to optimize ξ_* using example trajectories from the new environment e_*. While architecture-agnostic and sample-efficient at inference, these methods require an adaptation step that can be computationally expensive due to the need for gradient descent.

In-context learning Instead of relying on gradient-based adaptation, in-context learning methods use an encoder to condition a processor or decoder. ICON (Yang et al., 2023) is an encoder-decoder transformer trained to predict the PDE solution u_* given an input parameter a_* and a set of context examples $(a_1, u_1), (a_2, u_2), \ldots, (a_k, u_k)$. While effective for operator learning, its scalability is limited since the inputs are processed point-wise by the transformer. Cao et al. (2024) addressed this issue by introducing patch-based processing to scale the method for 2D temporal dynamics. However, this approach requires a history of frames and cannot be used with a single initial condition at inference.

Though not strictly in-context learning, Zhou and Farimani (2024) explored a masked autoencoder (He et al., 2021) pretraining strategy on PDE trajectories. They showed that this encoder could extract meaningful embeddings that could be used to condition classical surrogate models. However, their focus is not on adaptation. Similarly, Chen et al. (2024) proposed an unsupervised strategy to pretrain a neural operator using only initial conditions.

Multiple physics Inspired by the development of foundation models in vision and language modeling, another promising direction is to design neural networks capable of handling entirely different physical systems—such as both Navier-Stokes and Maxwell's equations—rather than just variations in PDE parameters. This concept,

known as Multiple Physics Pretraining, was introduced in McCabe et al. (2023), where a single network achieved state-of-the-art performance across PDEBench (Takamoto et al., 2022). Similar efforts with different architectures have since emerged (Herde et al., 2024; Hao et al., 2024).

However, these models require fine-tuning when only a single initial condition is available at inference, making them computationally costly and inefficient. Despite this limitation, they hold great potential for developing a true foundation model for physics. This promise is further strengthened by the recent release of high-quality datasets (Ohana et al., 2024), which could enable broader applications beyond direct surrogate modeling—such as extracting meaningful features for downstream tasks.

Part II

Contributions

Chapter 5

Operator Learning with Neural Fields: Tackling PDEs on General Geometries

In this chapter, we address the problem of operator learning with arbitrary geometries and discretizations. Machine learning approaches for solving partial differential equations require learning mappings between function spaces. While convolutional or graph neural networks are constrained to discretized functions, neural operators present a promising milestone toward mapping functions directly. Despite impressive results they still face challenges with respect to the domain geometry and typically rely on some form of discretization. In order to alleviate such limitations, we present CORAL, a new method that leverages coordinate-based networks for solving PDEs on general geometries. CORAL is designed to remove constraints on the input mesh, making it applicable to any spatial sampling and geometry. Its ability extends to diverse problem domains, including PDE solving, spatio-temporal forecasting, and geometry-aware inference. CORAL[1] demonstrates robust performance across multiple resolutions and performs well in both convex and non-convex domains, surpassing or performing on par with state-of-the-art models.

Serrano, L., Le Boudec, L., Kassaï Koupaï, A., Wang, T. X., Yin, Y., Vittaut, J. N., Gallinari, P. (2023). Operator learning with neural fields: Tackling pdes on general geometries. Neurips 2023.

[1]The source code for this paper is available at https://github.com/LouisSerrano/coral

5.1 Introduction

Modeling physics dynamics entails learning mappings between function spaces, a crucial step in formulating and solving partial differential equations (PDEs). In the classical approach, PDEs are derived from first principles, and differential operators are utilized to map vector fields across the variables involved in the problem. To solve these equations, numerical methods like finite elements, finite volumes, or spectral techniques are employed, requiring the discretization of spatial and temporal components of the differential operators (Morton and Mayers, 2005; Olver, 2014).

Building on successes in computer vision and natural language processing (Krizhevsky et al., 2017; He et al., 2016a; Dosovitskiy et al., 2021; Vaswani et al., 2017), deep learning models have recently gained attention in physical modeling. They have been applied to various scenarios, such as solving PDEs (Cai et al., 2021), forecasting spatio-temporal dynamics (Bézenac et al., 2017), and addressing inverse problems (Allen et al., 2022). Initially, neural network architectures with spatial inductive biases like ConvNets (Long et al., 2018; Ibrahim et al., 2022) for regular grids or GNNs (Pfaff et al., 2021; Brandstetter et al., 2022d) for irregular meshes were explored. However, these models are limited to specific mesh points and face challenges in generalizing to new topologies. The recent trend of neural operators addresses these limitations by modeling mappings between functions, which can be seen as infinite-dimensional vectors. Popular models like DeepONet (Lu et al., 2022) and Fourier Neural Operators (FNO) (Li et al., 2022b) have been applied in various domains. However, they still have design rigidity, relying on fixed grids during training and inference, which limits their use in real-world applications involving irregular sampling grids or new geometries. A variant of FNO tailored for more general geometries is presented in (Li et al., 2022a), but it focuses on design tasks.

To overcome these limitations, there is a need for flexible approaches that can handle diverse geometries, metric spaces, irregular sampling grids, and sparse mea-

surements. We introduce CORAL, a COordinate-based model for opeRAtor Learning that addresses these challenges by leveraging implicit neural representations (INR). CORAL encodes functions into compact, low-dimensional latent spaces and infers mappings between function representations in the latent space. Unlike competing models that are often task-specific, CORAL is highly flexible and applicable to various problem domains. We showcase its versatility in PDE solving, spatio-temporal dynamics forecasting, and design problems.

Our contributions are summarized as follows:

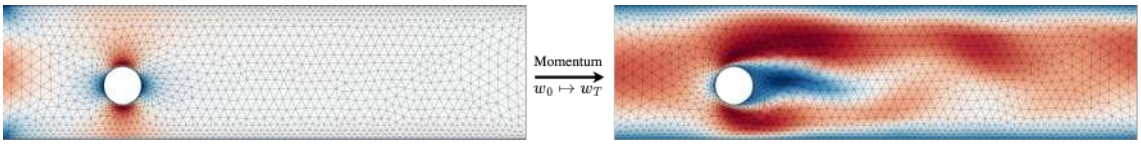
- CORAL can learn mappings between functions sampled on an irregular mesh and maintains consistent performance when applied to new grids not seen during training. This characteristic makes it well-suited for solving problems in domains characterized by complex geometries or non-uniform grids.
- We highlight the versatility of CORAL by applying it to a range of representative physical modeling tasks, such as initial value problems (IVP), geometry-aware inference, dynamics modeling, and forecasting. Through extensive experiments on diverse datasets, we consistently demonstrate its state-of-the-art performance across various geometries, including convex and non-convex domains, as well as planar and spherical surfaces. This distinguishes CORAL from alternative models that are often confined to specific tasks.
- CORAL is fast. Functions are represented using a compact latent code in CORAL, capturing the essential information necessary for different inference tasks in a condensed format. This enables fast inference within the compact representation space, whereas alternative methods often operate directly within a higher-dimensional representation of the function space.

5.2 Related Work

Mesh-based networks for physics. The initial attempts to learn physical dynamics primarily centered around convolutional neural networks (CNNs) and graph neural networks (GNNs). Both leverage discrete convolutions to extract relevant information from a given node's neighborhood Hamilton (2020). CNNs expect inputs and outputs to be on regular grid. Their adaptation to irregular data through interpolation (Chae et al., 2021) is limited to simple meshes. GNNs work on irregular meshes (Hamilton et al., 2017b; Veličković et al., 2018; Pfaff et al., 2021) and have been used e.g. for dynamics modeling (Brandstetter et al., 2022d) or design optimization (Allen et al., 2022). They typically select nearest neighbors within a small radius, which can introduce biases towards the type of meshes seen during training. In Section 5.4.2, we show that this bias can hinder their ability to generalize to meshes with different node locations or levels of sparsity. Additionally,

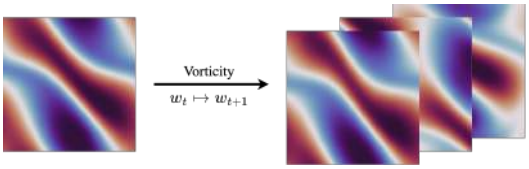
they require significantly more memory resources than plain CNNs to store nodes' neighborhoods, which limits their deployment for complex meshes.

Initial Value Problem

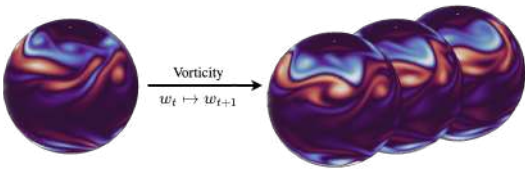


(a) *Cylinder*

Dynamics modeling

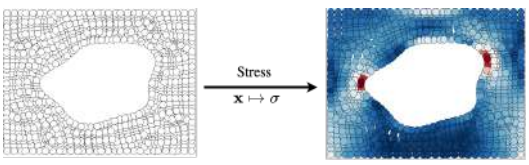


(b) *Navier-Stokes*

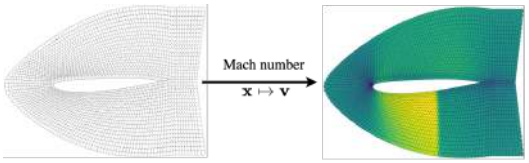


(c) *Shallow-Water*

Geometry-aware inference



(d) *Elasticity*



(e) *NACA-Euler*

Figure 5.1: Illustration of the problem classes addressed in this work: Initial Value Problem (IVP) (a), dynamic forecasting (b and c) and geometry-aware inference (d and e).

Operator learning. Operator learning is a burgeoning field in deep learning for physics that focuses on learning mappings between infinite-dimensional functions. Two prominent approaches are DeepONet (Lu et al., 2021a) and Fourier Neural Operator (FNO; Li et al., 2021). DeepONet can query any coordinate in the domain for a value of the output function. However, the input function must be observed on a set of predefined locations, requiring the same observation grid for all observations, for training and testing. FNO is an instance of neural operators (Kovachki et al., 2021), a family of approaches that integrate kernels over the spatial domain. Since this operation can be expensive, FNO addresses the problem by employing the fast Fourier transform (FFT) to transform the inputs into the spectral domain. As a consequence it cannot be used with irregular grids. Li et al. (2022a) introduce an FNO extension to handle more flexible geometries, but it is tailored for design problems. To summarize, despite promising for several applications, current operator approaches still face limitations to extrapolate to new geometries; they do not adapt to changing observation grids or are limited to fixed observation locations. Recently, Li et al. (2023a); Hao et al. (2023) explored transformer-based

architectures as an alternative approach.

Spatial INRs. Spatial INRs are a class of coordinate-based neural networks that model data as the realization of an implicit function of a spatial location $x \in \Omega \mapsto f_\theta(x)$ (Tancik et al., 2020b; Sitzmann et al., 2020c; Fathony et al., 2021b; Lindell et al., 2022). An INR can be queried at any location, but encodes only one data sample or function. Previous works use meta-learning (Tancik et al., 2021; Sitzmann et al., 2020a), auto-encoders (Chen and Zhang, 2019; Mescheder et al., 2019), or modulation (Park et al., 2019a; Dupont et al., 2022a) to address this limitation by enabling an INR to decode various functions using per-sample parameters. INRs have started to gain traction in physics, where they have been successfully applied to spatio-temporal forecasting (Yin et al., 2022b) and reduced-order modeling (Chen et al., 2022). The former work is probably the closest to ours but it is designed for forecasting and cannot handle the range of tasks that CORAL can address. Moreover, its computational cost is significantly higher than CORAL's, which limits its application in real-world problems. The work by Chen et al. (2022) aims to inform the INR with known PDEs, similar to PINNs, whereas our approach is entirely data-driven and without physical prior.

5.3 The CORAL Framework

In this section, we present the CORAL framework, a novel approach that employs an encode-process-decode structure to achieve the mapping between continuous functions. We first introduce the model and then the training procedure.

5.3.1 Problem Description

Let $\Omega \subset \mathbb{R}^d$ be a bounded open set of spatial coordinates. We assume the existence of a mapping $\mathcal{G}^*$ from one infinite-dimensional space $\mathcal{A} \subset L^2(\Omega, \mathbb{R}^{d_a})$ to another one $\mathcal{U} \subset L^2(\Omega, \mathbb{R}^{d_u})$, such that for any observed pairs $(a_i, u_i) \in \mathcal{A} \times \mathcal{U}$, $u_i = \mathcal{G}^*(a_i)$. We have $a_i \sim \nu_a$, $u_i \sim \nu_u$ where ν_a is a probability measure supported on $\mathcal{A}$ and ν_u the pushforward measure of ν_a by $\mathcal{G}^*$. We seek to approximate this operator by an i.i.d. collection of point-wise evaluations of input-output functions through a highly flexible formulation that can be adapted to multiple tasks. In this work, we target three different tasks as examples: • solving an initial value problem, i.e. mapping the initial condition $u_0 \doteq x \mapsto u(x, t = 0)$ to the solution at a predefined time $u_T \doteq x \mapsto u(x, t = T)$, • modeling the dynamics of a physical system over time $(u_t \to u_{t+\Delta t})$ over a given forecasting horizon • or prediction based on geometric configuration. At training time, we have access to n_{tr} pairs of input and output functions $(a_i, u_i)_{i=1}^{n_{tr}}$ evaluated over a free-form spatial grid $\mathcal{X}_i$. We denote $a|_{\mathcal{X}_i} = (a(x))_{x \in \mathcal{X}_i}$ and $u|_{\mathcal{X}_i} = (u(x))_{x \in \mathcal{X}_i}$ the vectors of the function values over the

sample grid. In the context of the initial value and geometry-aware problems, every sample is observed on a specific grid $\mathcal{X}_i$. For dynamics modeling, we use a unique grid $\mathcal{X}_{tr}$ for all the examples to train the model and another grid $\mathcal{X}_{te}$ for testing.

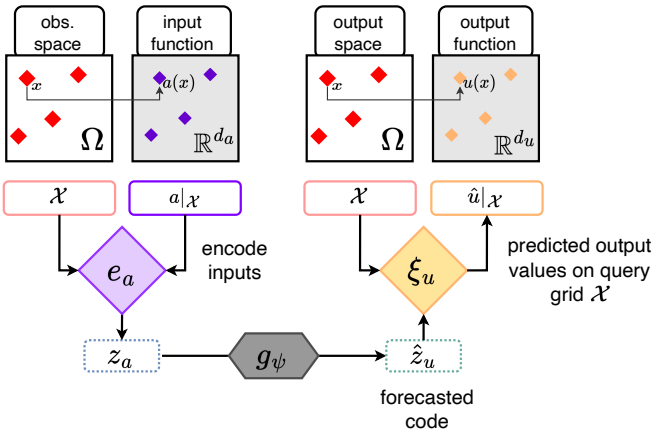


Figure 5.2: Inference for CORAL. First, the model embeds the input function a without constraints on the locations of the observed sensors into an input latent code z_a, then infers the output latent code $\hat{z}_u$ and finally predicts the output value $\hat{u}(x)$ for any query coordinate $x \in \Omega$. For the grid $\mathcal{X}$, we use the vector notation $a|_{\mathcal{X}} = (a(x))_{x\in\mathcal{X}}$, $\hat{u}|_{\mathcal{X}} = (\hat{u}(x))_{x\in\mathcal{X}}$.

5.3.2 Model

CORAL makes use of two modulated INRs, f_{θ_a,ϕ_a} and f_{θ_u,ϕ_u}, for respectively representing the input and output functions of an operator. While θ_a and θ_u denote shared INR parameters that contribute in representing all functions a_i and u_i, the modulation parameters ϕ_{a_i} and ϕ_{u_i} are specific to each function a_i and u_i. Given input/output INR functions representation, CORAL then learns a mapping between latent representations inferred from the two INRs' modulation spaces. The latent representations z_{a_i}, z_{u_i} are low dimensional embeddings, capturing within a compact code information from the INRs' parameters. They are used as inputs to hypernetworks h_a and h_u to compute the modulation parameters $\phi_{a_i} = h_a(z_{a_i})$ and $\phi_{u_i} = h_u(z_{u_i})$. The weights of the input and output hypernetworks are respectively denoted w_a and w_u.

CORAL proceeds in three steps: *encode*, to project the input data into the latent space; *process*, to perform transformations in the latent space; and *decode*, to project the code back to the output function space. First, the input function a is encoded into the small input latent code z_a using a spatial encoder $e_a : \mathcal{A} \mapsto \mathbb{R}^{d_z}$. Next, a parameterized model $g_\psi : \mathbb{R}^{d_z} \mapsto \mathbb{R}^{d_z}$ is used to infer an output latent code. Depending on the target task, g_ψ can be as simple as a plain MLP or more complex as for example a neural ODE solver (as detailed later). Finally, the processed latent

code is decoded into a spatial function using a decoder $\xi_u : \mathbb{R}^{d_z} \mapsto \mathcal{U}$. The resulting CORAL operator then writes as $\mathcal{G} = \xi_u \circ g_\psi \circ e_a$, as shown in Figure 5.2. The three steps are detailed below.

Encode Given an input function a_i and a learned shared parameter θ_a, the encoding process provides a code $z_{a_i} = e_a(a_i)$. This code is computed by solving an inverse problem through a procedure known as *auto-decoding*, which proceeds as follows. We want to compress into a compact code z_{a_i} the information required for reconstructing the original field a_i through the input INR, i.e.: $\forall x \in \mathcal{X}_i, f_{\theta_a,\phi_{a_i}}(x) = \tilde{a}_i(x) \approx a_i(x)$ with $\phi_{a_i} = h_a(z_{a_i})$. See Figure A.1a in Appendix A.2 for details. The approximate solution to this inverse problem is computed as the solution $e_a(a_i) = z_{a_i}^{(K)}$ of a gradient descent optimization:

$$z_{a_i}^{(0)} = 0 \; ; z_{a_i}^{(k+1)} = z_{a_i}^{(k)} - \alpha \nabla_{z_{a_i}^{(k)}} \mathcal{L}_{\mu_i}(f_{\theta_a,\phi_{a_i}^{(k)}}, a); \text{ with } \phi_{a_i}^{(k)} = h_a(z_{a_i}^{(k)}) \text{ for } 0 \leq k \leq K-1 \tag{5.1}$$

where α is the inner loop learning rate, K the number of inner steps, and $\mathcal{L}_{\mu_i}(v, w) = \mathbb{E}_{x\sim\mu_i}[(v(x) - w(x))^2]$ for a measure μ_i over Ω. Note that in practice, μ_i is defined through the observation grid $\mathcal{X}_i$, $\mu_i(\cdot) = \sum_{x\in\mathcal{X}_i} \delta_x(\cdot)$ where $\delta_x(\cdot)$ is the Dirac measure. Since we can query the INRs anywhere within the domain, we can hence freely encode functions without mesh constraints. This is the essential part of the architecture that enables us to feed data defined on different grids to the model. We show the encoding flow in Appendix A.2, Figure A.2.

Process Once we obtain z_{a_i}, we can infer the latent output code $\hat{z}_{u_i} = g_\psi(z_{a_i})$. For simplification, we consider first that g_ψ is implemented through an MLP with parameters ψ. For dynamics modeling, in Section 5.4.2, we will detail why and how to make use of a Neural ODE solver for g_ψ.

Decode We decode $\hat{z}_{u_i}$ with the output hypernetwork h_u and modulated INR and denote ξ_u the mapping that associates to code $\hat{z}_{u_i}$ the function $f_{\theta_u,\hat{\phi}_{u_i}}$, where $\hat{\phi}_{u_i} = h_u(\hat{z}_{u_i})$. Since $f_{\theta_u,\hat{\phi}_{u_i}}$ is an INR, i.e. a function of spatial coordinates, it can be freely queried at any point within the domain. We thus have $\forall x \in \Omega, \hat{u}_i(x) = \xi_u(\hat{z}_{u_i})(x) = f_{\theta_u,\hat{\phi}_{u_i}}(x)$. See Figure A.1b in Appendix A.2 for details.

During training, we will need to learn to reconstruct the input and output functions a_i and u_i. This requires training a mapping associating an input code to the corresponding input function $\xi_a : \mathbb{R}^{d_z} \mapsto \mathcal{A}$ and a mapping associating a function to its code in the output space $e_u : \mathcal{U} \mapsto \mathbb{R}^{d_z}$, even though they are not used during inference.

5.3.3 Practical implementation: decoding by INR Modulation

We choose SIREN (Sitzmann et al., 2020c) – a state-of-the-art coordinate-based network – as the INR backbone of our framework. SIREN is a neural network that uses sine activations with a specific initialization scheme (Appendix A.2).

$$f_\theta(x) = \boldsymbol{W}_L\big(\sigma_{L-1} \circ \sigma_{L-2} \circ \cdots \circ \sigma_0(x)\big) + \boldsymbol{b}_L, \text{with } \sigma_i(\eta_i) = \sin\big(\omega_0(\boldsymbol{W}_i\eta_i + \boldsymbol{b}_i)\big) \quad (5.2)$$

where $\eta_0 = x$ and $(\eta_i)_{i\geq 1}$ are the hidden activations throughout the network. $\omega_0 \in \mathbb{R}_+^*$ is a hyperparameter that controls the frequency bandwidth of the network, $\boldsymbol{W}$ and $\boldsymbol{b}$ are the network weights and biases. We implement shift modulations (Perez et al., 2018b) to have a small modulation space and reduce the computational cost of the overall architecture. This yields the modulated SIREN:

$$f_{\theta,\phi}(x) = \boldsymbol{W}_L\big(\sigma_{L-1} \circ \sigma_{L-2} \circ \cdots \circ \sigma_0(x)\big) + \boldsymbol{b}_L, \text{with } \sigma_i(\eta_i) = \sin\big(\omega_0(\boldsymbol{W}_i\eta_i + \boldsymbol{b}_i + \boldsymbol{\phi}_i)\big) \quad (5.3)$$

with shared parameters $\theta = (\boldsymbol{W}_i, \boldsymbol{b}_i)_{i=0}^L$ and example associated modulations $\phi = (\boldsymbol{\phi}_i)_{i=0}^{L-1}$. We compute the modulations ϕ from z with a linear hypernetwork , i.e. for $0 \leq i \leq L-1$, $\boldsymbol{\phi}_i = \boldsymbol{V}_i z + \boldsymbol{c}_i$. The weights $\boldsymbol{V}_i$ and $\boldsymbol{c}_i$ constitute the parameters of the hypernetwork $w = (\boldsymbol{V}_i, \boldsymbol{c}_i)_{i=0}^{L-1}$. This implementation is similar to that of Dupont et al. (2022a), which use a modulated SIREN for representing their modalities.

5.3.4 Training

We implement a two-step training procedure that first learns the modulated INR parameters, before training the forecast model g_ψ. It is very stable and much faster than end-to-end training while providing similar performance: once the input and output INRs have been fitted, the training of g_ψ is performed in the small dimensional modulated INR z-code space. Formally, the optimization problem is defined as:

$$\begin{aligned} &\arg\min_{\psi} \mathbb{E}_{a,u\sim\nu_a,\nu_u} \|g_\psi(\tilde{e}_a(a)) - \tilde{e}_u(u)\|^2 \\ \text{s.t. } \tilde{e}_a &= \arg\min_{\xi_a, e_a} \mathbb{E}_{a\sim\nu_a} \mathcal{L}(\xi_a \circ e_a(a), a) \\ \text{and } \tilde{e}_u &= \arg\min_{\xi_u, e_u} \mathbb{E}_{u\sim\nu_u} \mathcal{L}(\xi_u \circ e_u(u), u) \end{aligned} \quad (5.4)$$

Note that functions (e_u, ξ_u) and (e_a, ξ_a) are parameterized respectively by the weights (θ_u, w_u) and (θ_a, w_a), of the INRs and of the hypernetworks. In Equation (5.4), we used the (e_u, ξ_u) & (e_a, ξ_a) description for clarity, but as they are functions of (θ_u, w_u) & (θ_a, w_a), optimization is tackled on the latter parameters. We outline the training pipeline in Appendix A.2, Figure A.3. During training, we constrain e_u, e_a to take only a few steps of gradient descent to facilitate the proces-

sor task. This regularization prevents the architecture from memorizing the training set into the individual codes and facilitates the auto-decoding optimization process for new inputs. In order to obtain a network that is capable of quickly encoding new physical inputs, we employ a second-order meta-learning training algorithm based on CAVIA (Zintgraf et al., 2019b). Compared to a first-order scheme such as Reptile (Nichol et al., 2018), the outer loop back-propagates the gradient through the K inner steps, consuming more memory as we need to compute gradients of gradients but yielding higher reconstruction results with the modulated SIREN. We experimentally found that using 3 inner-steps for training, or testing, was sufficient to obtain very low reconstruction errors for most applications.

5.4 Experiments

To demonstrate the versatility of our model, we conducted experiments on three distinct tasks (Figure 5.1): (i) solving an initial value problem (Section 5.4.1), (ii) modeling the dynamics of a physical system (Section 5.4.2), and (iii) learning to infer the steady state of a system based on the domain geometry (Section 5.4.3) plus an associated design problem in Appendix A.4. Since each task corresponds to a different scenario, we utilized task-specific datasets and employed different baselines for each task. This approach was necessary because existing baselines typically focus on specific tasks and do not cover the full range of problems addressed in our study, unlike CORAL. We provide below an introduction to the datasets, evaluation protocols, and baselines for each task setting. All experiments were conducted on a single GPU: NVIDIA RTX A5000 with 25 Go. The code is accessible at this github repository: `https://github.com/LouisSerrano/coral`.

5.4.1 Initial Value Problem

An IVP is specified by an initial condition (here the input function providing the state variables at $t = 0$) and a target function figuring the state variables value at a given time T. Solving an IVP is a direct application of the CORAL framework introduced in Section 5.3.2.

Datasets We benchmark our model on two problems with non-convex domains proposed in Pfaff et al. (2021). In both cases, the fluid evolves in a domain – which includes an obstacle – that is more densely discretized near the boundary conditions (BC). The boundary conditions are provided by the mesh definition, and the models are trained on multiple obstacles and evaluated at test time on similar but different obstacles. • **Cylinder** simulates the flow of water around a cylinder on a fixed 2D Eulerian mesh, and is characteristic of *incompressible* fluids. For each node j we have access to the node position $x^{(j)}$, the momentum $w(x^{(j)})$ and the pressure

$p(x^{(j)})$. We seek to learn the mapping $(x, w_0(x), p_0(x))_{x\in\mathcal{X}} \rightarrow (w_T(x), p_T(x))_{x\in\mathcal{X}}$. • **Airfoil** simulates the aerodynamics around the cross-section of an airfoil wing, and is an important use-case for *compressible* fluids. In this dataset, we have in addition for each node j the fluid density $\rho(x^{(j)})$, and we seek to learn the mapping $(x, w_0(x), p_0(x), \rho_0(x))_{x\in\mathcal{X}} \rightarrow (w_T(x), p_T(x), \rho_T(x))_{x\in\mathcal{X}}$. For both datasets, each example is associated to a mesh and the meshes are different for each example. For *Airfoil* the average number of nodes per mesh is 5233 and for *Cylinder* 1885.

Evaluation protocols Training is performed using all the mesh points associated to an example. For testing we evaluate the following two settings. • **Full**, we validate that the trained model generalizes well to new examples using all the mesh location points of these examples. • **Sparse** We assess the capability of our model to generalize on sparse meshes: the original input mesh is down-sampled by randomly selecting 20% of its nodes. We use a train, validation, test split of 1000 / 100 / 100 samples for all the evaluations.

Baselines We compare our model to • **NodeMLP**, a FeedForward Neural Network that ignores the node neighbors and only learns a local mapping • **GraphSAGE** (Hamilton et al., 2017b), a popular GNN architecture that uses SAGE convolutions • **MP-PDE** (Brandstetter et al., 2022d), a message passing GNN that builds on (Pfaff et al., 2021) for solving PDEs.

Results. We show in Table 5.1 the performance on the test sets for the two datasets and for both evaluation settings. Overall, CORAL is on par with the best models for this task. For the *Full* setting, it is best on *Cylinder* and second on *Airfoil* behind MP-PDE. However, for the *sparse* protocol, it can infer the values on the full mesh with the lowest error compared to all other models. Note that this second setting is more challenging for *Cylinder* than for *Airfoil* given their respective average mesh size. This suggests that the interpolation of the model outputs is more robust on the *Airfoil* dataset, and explains why the performance of NodeMLP remains stable between the two settings. While MP-PDE is close to CORAL in the *sparse* setting, GraphSAGE fails to generalize, obtaining worse predictions than the local model. This is because the model aggregates neighborhood information regardless of the distance between nodes, while MP-PDE does consider node distance and difference between features.

Table 5.1: Initial Value Problem - Test results. MSE on normalized data.

Model	*Cylinder*		*Airfoil*	
	Full	*Sparse*	*Full*	*Sparse*
NodeMLP	1.48e-1 ± 2.00e-3	2.29e-1 ± 3.06e-3	2.88e-1 ± 1.08e-2	2.83e-1 ± 2.12e-3
GraphSAGE	7.40e-2 ± 2.22e-3	2.66e-1 ± 5.03e-3	2.47e-1 ± 7.23e-3	5.55e-1 ± 5.54e-2
MP-PDE	8,72e-2 ± 4.65e-3	1.84e-1 ± 4.58e-3	**1.97e-1 ± 1.34e-2**	3.07e-1 ± 2.56e-2
CORAL	**7.03e-2 ± 5.96e-3**	**1.70e-1 ± 2.53e-2**	2.40e-1 ± 4.36e-3	**2.43e-1 ± 4.14e-3**

5.4.2 Dynamics Modeling

For the IVP problem, in Section 5.4.1, the objective was to infer directly the state of the system at a given time T given an initial condition (IC). We can extend this idea to model the dynamics of a physical system over time, so as to forecast state values over a given horizon. We have developed an autoregressive approach operating on the latent code space for this problem. Let us denote $(u_0, u_{\Delta t}, ..., u_{L\Delta t})$ a target sequence of observed functions of size $L+1$. Our objective will be to predict the functions $u_{k\Delta t}, k = 1, ..., L$, starting from an initial condition u_0. For that we will encode $z_0 = e(u_0)$, then predict sequentially the latent codes $z_{k\Delta t}, k = 1, ..., L$ using the processor in an auto regressive manner, and decode the successive values to get the predicted $\hat{u}_{k\Delta t}, k = 1, ..., L$ at the successive time steps.

5.4.2.1 Implementation with Neural ODE

The autoregressive processor is implemented by a Neural ODE solver operating in the latent z-code space. Compared to the plain MLP implementation used for the IVP task, this provides both a natural autoregressive formulation, and overall, an increased flexibility by allowing to forecast at any time in a sequence, including different time steps or irregular time steps. Starting from any latent state z_t, a neural solver predicts state $z_{t+\tau}$ as $z_{t+\tau} = z_t + \int_t^{t+\tau} \zeta_\psi(z_s)ds$ with ζ_ψ a neural network with parameters to be learned, for any time step τ. The autoregressive setting directly follows from this formulation. Starting from z_0, and specifying a series of forecast time steps $k\Delta t$ for $k = 1, ..., L$, the solver call $\text{NODESolve}(\zeta_\psi, z_0, \{k\Delta t\}_{k=1,...,L})$ will compute predictions $z_{k\Delta t}, k = 1, ..., L$ autoregressively, i.e. using $z_{k\Delta t}$ as a starting point for computing $z_{(k+1)\Delta t}$. In our experiments we have used a fourth-order Runge-Kutta scheme (RK4) for solving the integral term. Therefore, the predicted field at time step k can be obtained as $\hat{u}_{k\Delta t} = \xi \circ \text{NODESolve}(\zeta_\psi, z_0 = e(u_0), k\Delta t)$. Note that when solving the IVP problem from Section 5.4.1, two INRs are used, one for encoding the input function and one for the output function; here a single modulated INR $f_{\theta,\phi}$ is used to represent a physical quantity throughout the sequence at any time. θ is then shared by all the elements of a sequence and ϕ is computed by a hypernetwork to produce a function specific code.

We use the two-step training procedure from Section 5.3.4, i.e. first the INR is trained to auto-decode the states of each training trajectory, and then the processor operating over the codes is learned through a Neural ODE solver according to Equation (5.5).

$$\arg\min_{\psi} \mathbb{E}_{u\sim\nu_u, t\sim\mathcal{U}(0,T]} \|g_\psi(\tilde{e}_u(u_0), t) - \tilde{e}_u(u_t)\|^2 \tag{5.5}$$

The two training steps are separated and the codes are kept fixed during the second step. This allows for a fast training as the Neural ODE solver operates on

the low dimensional code embedding space.

5.4.2.2 Experiment details

Datasets We consider two fluid dynamics equations for generating the datasets and refer the reader to Appendix A.1 for additional details. • **2D-Navier-Stokes equation** (*Navier-Stokes*) for a viscous, incompressible fluid in vorticity form on the unit torus: $\frac{\partial w}{\partial t} + u \cdot \nabla w = \nu \Delta w + f$, $\nabla u = 0$ for $x \in \Omega, t > 0$, where $\nu = 10^{-3}$ is the viscosity coefficient. The train and test sets are composed of 256 and 16 trajectories respectively where we observe the vorticity field for 40 timestamps. The original spatial resolution is 256×256 and we sub-sample the data to obtain frames of size 64×64. • **3D-Spherical Shallow-Water equation** (*Shallow-Water*) can be used as an approximation to a flow on the earth's surface. The data consists of the vorticity w, and height h of the fluid. The train and test sets are composed respectively of 16 and 2 long trajectories, where we observe the vorticity and height fields for 160 timestamps. The original spatial resolution is 128 (lat) $\times$ 256 (long), which we sub-sample to obtain frames of shape 64×128. We model the dynamics with the complete state (h, w). Each trajectory, for both datasets and for train and test is generated from a different initial condition (IC).

Setting We evaluate the ability of the model to generalize in space and time. • **Temporal extrapolation:** For both datasets, we consider sub-trajectories of 40 timestamps that we split in two equal parts of size 20, with the first half denoted *In-t* and the second one *Out-t*. The training-*In-t* set is used to train the models at forecasting the horizon $t = 1$ to $t = 19$. At test time, we unroll the dynamics from a new IC until $t = 39$. Evaluation in the horizon *In-t* assesses CORAL's capacity to forecast within the training horizon. *Out-t* allows evaluation beyond *In-t*, from $t = 20$ to $t = 39$. • **Varying sub-sampling:** We randomly sub-sample π percent of a regular mesh to obtain the train grid $\mathcal{X}_{tr}$, and a second test grid $\mathcal{X}_{te}$, that are shared across trajectories. The train and test grids are different, but have the same level of sparsity. • **Up-sampling:** We also evaluate the up-sampling capabilities of CORAL in Appendix A.3. In these experiments, we trained the model on a sparse, low-resolution grid and evaluate its performance on high resolution-grids.

Baselines To assess the performance of CORAL, we implement several baselines: two operator learning models, one mesh-based network and one coordinate-based method. • **DeepONet** (Lu et al., 2021a): we train DeepONet in an auto-regressive manner with time removed from the trunk net's input. • **FNO** (Li et al., 2021): we use an auto-regressive version of the Fourier Neural Operator. • **MP-PDE** (Brandstetter et al., 2022d) : we use MP-PDE as the irregular mesh-based baseline. We fix MP-PDE's temporal bundling to 1, and train the model with the push-forward trick. • **DINo** (Yin et al., 2022b) : We finally compare CORAL with DINo, an

INR-based model designed for dynamics modeling.

5.4.2.3 Results

Table 5.2: Temporal Extrapolation - Test results. Metrics in MSE.

$\times\mathcal{X}_{tr} \downarrow \mathcal{X}_{te}$	dataset →	*Navier-Stokes*		*Shallow-Water*	
		In-t	*Out-t*	*In-t*	*Out-t*
$\pi = 100\%$ regular grid	DeepONet	4.72e-2 ± 2.84e-2	9.58e-2 ± 1.83e-2	6.54e-3 ± 4.94e-4	8.93e-3 ± 9.42e-5
	FNO	5.68e-4 ± 7.62e-5	8.95e-3 ± 1.50e-3	3.20e-5 ± 2.51e-5	**1.17e-4 ± 3.01e-5**
	MP-PDE	4.39e-4 ± 8.78e-5	4.46e-3 ± 1.28e-3	9.37e-5 ± 5.56e-6	1.53e-3 ± 2.62e-4
	DINo	1.27e-3 ± 2.22e-5	1.11e-2 ± 2.28e-3	4.48e-5 ± 2.74e-6	2.63e-3 ± 1.36e-4
	CORAL	**1.86e-4 ± 1.44e-5**	**1.02e-3 ± 8.62e-5**	**3.44e-6 ± 4.01e-7**	4.82e-4 ± 5.16e-5
$\pi = 20\%$ irregular grid	DeepONet	8.37e-1 ± 2.07e-2	7.80e-1 ± 2.36e-2	1.05e-2 ± 5.01e-4	1.09e-2 ± 6.16e-4
	FNO + lin. int.	3.97e-3 ± 8.03e-4	9.92e-3 ± 2.36e-3	n.a.	n.a.
	MP-PDE	3,98e-2 ± 1,69e-2	1,31e-1 ± 5,34e-2	5.28e-3 ± 5.25e-4	2.56e-2 ± 8.23e-3
	DINo	**9.99e-4 ± 6.71e-3**	8.27e-3 ± 5.61e-3	2.20e-3 ± 1.06e-4	4.94e-3 ± 1.92e-4
	CORAL	2.18e-3 ± 6.88e-4	**6.67e-3 ± 2.01e-3**	**1.41e-3 ± 1.39e-4**	**2.11e-3 ± 5.58e-5**
$\pi = 5\%$ irregular grid	DeepONet	7.86e-1 ± 5.48e-2	7.48e-1 ± 2.76e-2	1.11e-2 ± 6.94e-4	**1.12e-2 ± 7.79e-4**
	FNO + lin. int.	3.87e-2 ± 1.44e-2	5.19e-2 ± 1.10e-2	n.a.	n.a.
	MP-PDE	1.92e-1 ± 9.27e-2	4.73e-1 ± 2.17e-1	1.10e-2 ± 4.23e-3	4.94e-2 ± 2.36e-2
	DINo	8.65e-2 ± 1.16e-2	9.36e-2 ± 9.34e-3	**1.22e-3 ± 2.05e-4**	1.52e-2 ± 3.74e-4
	CORAL	**2.44e-2 ± 1.96e-2**	**4.57e-2 ± 1.78e-2**	8.77e-3 ± 7.20e-4	1.29e-2 ± 1.92e-3

Table 5.2 details the performance of the different models in a combined temporal and spatial evaluation setting. • **General remarks:** CORAL demonstrates strong performance across all scenarios for both datasets. Only DINo exhibits similar properties, i.e., stability across spatial subsamplings and extrapolation horizon. We observe that all models performance degrade with lower sampling ratio. Also, as the models have been trained only on *In-t* horizon, error accumulates over time and thus leads to lower performance for *Out-t* evaluation. • **Analysis per model:** Although achieving strong performance on some specific scenarios, DeepONet, FNO and MP-PDE results are dependent of the training grid, geometries or number of points. FNO, can only be trained and evaluated on regular grids while DeepONet is not designed to be evaluated on a different grid in the branch net. MP-PDE achieves strong performance with enough sample positions, e.g. full grids here, but struggles to compete on irregular grids scenarios in Navier-Stokes. • **Inference Time:** We report in Appendix A.3, the inference time of the baselines considered. Despite operator methods have better inference time, CORAL is faster than mesh-free methods like DINo and MP-PDE. • **Generalization across samplings:** Coordinate-based methods demonstrate robustness when it comes to changes in spatial resolution. In contrast, MP-PDE model exhibits strong overfitting to the training grid, resulting in a decline in performance. Although MP-PDE and DINo may outperform CORAL in some settings, when changing the grid, CORAL remains stable and outperforms the other models. See Appendix A.3 for details.

5.4.3 Geometry-aware inference

In this section, we wish to infer the steady state of a system from its domain geometry, all other parameters being equal. The domain geometry is partially observed from the data in the form of point clouds or of a structured mesh $\mathcal{X}_i \subset \Omega_i$. The position of the nodes depends on the particular object shape. Each mesh $\mathcal{X}_i$ is obtained by deforming a reference grid $\mathcal{X}$ to adjust to the shape of the sample object. This grid deformation is the input function of the operator learning setting, while the output function is the physical quantity u_i over the domain Ω_i. The task objective is to train a model so as to generalize to new geometries, e.g. a new airfoil shape. Once a surrogate model has been trained to learn the influence of the domain geometry on the steady state solution, it can be used to quickly evaluate a new design and to solve inverse design problems (details in Appendix A.4).

Datasets. We used datasets generated from three different equations by Li et al. (2022a) and provide more details in Appendix A.1. • **Euler equation** (*NACA-Euler*) for a transonic flow over a NACA-airfoil. The measured quantity at each node is the Mach number. • **Navier-Stokes Equation** (*Pipe*) for an incompressible flow in a pipe, expressed in velocity form. The measured quantity at each node is the horizontal velocity. • **Hyper-elastic material** (*Elasticity*). Each sample represents a solid body with a void in the center of arbitrary shape, on which a tension is applied at the top. The material is the incompressible Rivlin-Saunders material and the measured quantity is the stress value. We use 1000 samples for training and 200 for test with all datasets.

Baselines We use • **Geo-FNO** (Li et al., 2022a) and • **Factorized-FNO** (Tran et al., 2023) two SOTA models as the main baselines. We also compare our model to regular-grid methods such as • **FNO** (Li et al., 2021) and • **UNet** (Ronneberger et al., 2015a), for which we first interpolate the input.

Table 5.3: Geometry aware inference - Test results. Relative L2 error.

Model	*NACA-Euler*	*Elasticity*	*Pipe*
FNO	3.85e-2 ± 3.15e-3	4.95e-2 ± 1.21e-3	1.53e-2 ± 8.19e-3
UNet	5.05e-2 ± 1.25e-3	5.34e-2 ± 2.89e-4	2.98e-2 ± 1.08e-2
Geo-FNO	1.58e-2 ± 1.77e-3	3.41e-2 ± 1.93e-2	**6.59e-3 ± 4.67e-4**
Factorized-FNO	6.20e-3 ± 3.00e-4	1.96e-2 ± 2.00e-2	7.33e-3 ± 4.66e-4
CORAL	**5.90e-3 ± 1.00e-4**	**1.67e-2 ± 4.18e-4**	1,20e-2 ± 8.74e-4

Results In Table 5.3 we can see that CORAL achieves state-of-the-art results on *Airfoil* and *Elasticity*, with the lowest relative error among all models. It is slightly below Factorized-FNO and Geo-FNO on *Pipe*. One possible cause is that this dataset exhibits high frequency only along the vertical dimension, while SIREN might be better suited for isotropic frequencies. Through additional experiments, we

demonstrate in Appendix A.4, how CORAL can also be used for solving an inverse problem corresponding to a design task: optimize the airfoil geometry to minimize the drag over lift ratio. This additional task further highlights the versatility of this model.

5.5 Discussion and limitations

Although a versatile model, CORAL inherits the limitations of INRs concerning the training time and representation power. It is then faster to train than GNNs, but slower than operators such as FNO, DeepONet and of course CNNs which might limit large scale deployments. Also some physical phenomena might not be represented via INRs. Although this is beyond the scope of this paper, it remains to evaluate the methods on large size practical problems. An interesting direction for future work would be to derive an efficient spatial latent representation for INRs, taking inspiration from grid-based representation for INRs (Takikawa et al. (2022b), Müller et al. (2022), Saragadam et al. (2022)). Another avenue would be to leverage clifford layers (Brandstetter et al., 2022a) to model interactions between physical fields.

5.6 Conclusion

We have presented CORAL, a novel approach for Operator Learning that removes constraints on the input-output mesh. CORAL offers the flexibility to handle spatial sampling or geometry variations, making it applicable to a wide range of scenarios. Through comprehensive evaluations on diverse tasks, we have demonstrated that it consistently achieves state-of-the-art or competitive results compared to baseline methods. By leveraging compact latent codes to represent functions, it enables efficient and fast inference within a condensed representation space.

Chapter 6

Infinity: Neural Field Modeling for Reynolds-Averaged Navier-Stokes Equations

For numerical design, the development of efficient and accurate surrogate models is paramount. They allow us to approximate complex physical phenomena, thereby reducing the computational burden of direct numerical simulations. We propose INFINITY, a deep learning model that utilizes implicit neural representations (INRs) to address this challenge. Our framework encodes geometric information and physical fields into compact representations and learns a mapping between them to infer the physical fields. We use an airfoil design optimization problem as an example task and we evaluate our approach on the challenging AirfRANS dataset, which closely resembles real-world industrial use-cases. The experimental results demonstrate that our framework achieves state-of-the-art performance by accurately inferring physical fields throughout the volume and surface. Additionally we demonstrate its applicability in contexts such as design exploration and shape optimization: our model can correctly predict drag and lift coefficients while adhering to the equations.

Serrano, L., Migus, L., Yin, Y., Mazari, J. A., Gallinari, P. (2023). Infinity: Neural field modeling for reynolds-averaged navier-stokes equations. ICML 2023 SynS ML workshop.

6.1 Introduction and motivation

Numerical simulations are essential for analyzing systems governed by partial differential equations (PDEs) in fields like fluid dynamics and climate science. These simulations involve discretizing the domain and solving the equations using methods such as finite differences, finite elements, or finite volumes (Reddy, 2019; Grossmann et al., 2007; Eymard et al., 2000). Since direct numerical simulation (DNS) can be computationally expensive or intractable, it is crucial to develop computationally efficient yet accurate surrogate models to accelerate the design process. Surrogate modeling for industrial applications, however, poses several challenges. The meshes used in these applications are extensive, consisting of hundreds of thousands of cells, and they also exhibit unstructured data and involve multi-scale phenomena. A typical example is the design of airfoils which will be our application focus, although the ideas can be easily implemented for other design tasks. In this domain, a new costly simulation must be run for each mesh during the optimization process, leading to time-consuming processes. Additionally, the design process focuses on finding the optimal shape for an airfoil that minimizes the force required for flight. Experts typically maximize the lift-over-drag ratio by solving equations across the entire mesh, with particular emphasis on the surface where various multi-scale phenomena occur.

Recently, deep learning methods have emerged as promising approaches for constructing surrogate models. However, the progress in this field was initially hindered by the lack of evaluation datasets representative of real-world data. The machine learning community has begun to address this issue by developing benchmarks. In this work, we utilize the a recent AirfRANS dataset Bonnet et al. (2022), which aims to replicate real-world industrial scenarios. This comprehensive benchmark provides an evaluation framework to assess the capabilities of deep learning (DL) in modeling the two-dimensional incompressible steady-state Reynolds-Averaged Navier-Stokes (RANS) equations for airfoils. Additionally, this 2D dataset encompasses a wide range of airfoil shapes derived from NASA's early works (Cummings et al., 2015), various turbulence effects characterized by Reynolds numbers and different angles of attack.

The Navier-Stokes equations are widely used in fluid dynamics, and as a result, numerous neural network surrogates have been proposed for their modeling in different contexts. Initial attempts all relied on grid-based approaches such as convolutional Neural Networks (CNNs) (Um et al., 2020a; Thuerey et al., 2020; Mohan et al., 2020; Wandel et al., 2020; Obiols-Sales et al., 2020; Gupta et al., 2021; Wang

et al., 2020a). CNNs face challenges when dealing with the irregular meshes used in computational fluid dynamics (CFD). Graph Neural Networks (GNNs) have shown promise Pfaff et al. (2021) but they have limitations in terms of receptive field size and information propagation across distant nodes, especially for large meshes. Additionally, GNNs struggle when the mesh is too dense and cannot fit into the memory of GPUs, necessitating sub-sampling. This limitation restricts their application in contexts where large meshes with multi-scale phenomena are prevalent. Furthermore, the evaluation of the models has primarily focused on traditional machine learning scores, such as global error over the entire domain (a.k.a. *volume*), rather than more design-oriented scores, including local error in the surface area surrounding the airfoil (a.k.a. *surface*) and errors in the aerodynamic forces of interest, such as drag and lift.

Leveraging recent advances in implicit neural representations (INRs) (Sitzmann et al., 2020b; Mildenhall et al., 2021), which have shown successful applications in physics problems (Yin et al., 2023), we introduce INFINITY, a model that utilizes coordinate-based networks to encode geometric information and physical fields into concise representations. INFINITY establishes a mapping between variables representing the problem's geometry and the corresponding physical fields, within this representation space. It possesses several unique features: (i) it is robust to varying mesh sampling, allowing for adaptability to different geometries, (ii) it effectively captures multi-scale phenomena, resulting in state-of-the-art scores for both volume and surface evaluations, (iii) as a continuous surrogate model, it can be used to accelerate the evaluation of different meshes during the design process, leading to significant speed-up. Importantly, we verified that INFINITY's field predictions accurately produce the correct lift and drag forces clearly outperforming all the baselines.

6.2 Method

6.2.1 Problem setting

We aim at proposing a surrogate model for airfoil design optimization in scenarios where the amount of available training data is limited ($n_{tr} \leq 1000$). Each airfoil is associated with a domain Ω_i, which is linked to a specific geometry. Consequently, different meshes $\mathcal{X}_i$ are generated within each domain. The characterization of an airfoil involves defining boundary conditions on $\partial\Omega_i$ corresponding to the airfoil surface, which are discretized into a surface mesh $\mathcal{S}_i$.

The geometric inputs for our model include the following information: • *Node positions* $\boldsymbol{x}$ represent the coordinates of each node within the airfoil's domain. • *Distance function* $d(\boldsymbol{x})$ provides the distance from each node to the surface of the airfoil. • *Normal vectors of the mesh nodes on the airfoil surface* $\boldsymbol{n}(\boldsymbol{x}) = (n_x(\boldsymbol{x}), n_y(\boldsymbol{x}))$

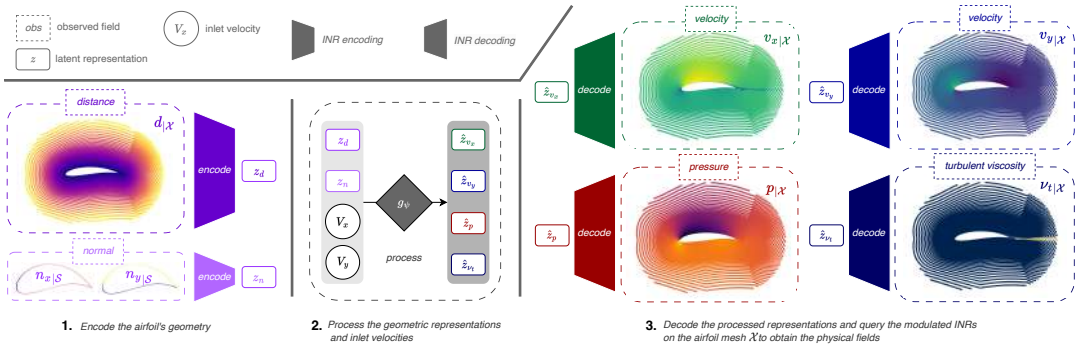


Figure 6.1: The inference of INFINITY proceeds in three steps. 1. We encode the distance function d and the normal components n_x, n_y into the latent representations z_d and z_n. 2. We process these codes along with the inlet velocities V_x, V_y to obtain the predicted output codes $\hat{z}_{v_x}, \hat{z}_{v_y}, \hat{z}_p, \hat{z}_{\nu_t}$ corresponding respectively to velocity, pressure and viscosity. 3. The processed codes are decoded with the modulated INRs, which can be queried directly at any mesh position $\boldsymbol{x} \in \mathcal{X}$.

specify the direction perpendicular to the airfoil surface at each node. In addition to the geometric inputs, we also have access to the inlet velocity values V_x and V_y, denoting the horizontal and vertical components of the velocity, respectively. It is worth noting that, on average, a mesh consists of approximately 200,000 nodes, providing a detailed representation of the airfoil's geometry.

The primary objective of the design optimization process is to maximize the lift-over-drag coefficient ratio, which serves as the key performance metric. To achieve this, we place significant emphasis on evaluating the relative errors in both the drag and lift coefficients, as well as assessing the Spearman correlation between predicted and actual values.

Rather than directly predicting the drag and lift values, our approach focuses on inferring various fluid fields associated with the airfoil's geometry. This includes calculating the velocities (v_x, v_y), pressure p, and turbulent kinematic viscosity ν_t on the mesh nodes, following the experimental protocol proposed in Bonnet et al. (2022). Therefore the inputs of our surrogate model are $(V_x, V_y, d|_{\mathcal{X}_i}, n_x|_{\mathcal{S}_i}, n_y|_{\mathcal{S}_i})_{i=1}^{n_{tr}}$, and the outputs are $(v_x|_{\mathcal{X}_i}, v_y|_{\mathcal{X}_i}, p|_{\mathcal{X}_i}, \nu_t|_{\mathcal{X}_i})_{i=1}^{n_{tr}}$. The output physical fields provide valuable insights into the underlying behavior of the fluid and its interaction with the airfoil's geometry. The drag and lift coefficients are calculated based on the predictions of the trained model while respecting the form of the RANS equations. This approach enables us to obtain a comprehensive understanding of the underlying fluid behavior and its relationship with the airfoil's geometry, thereby ultimately enhancing the accuracy of drag and lift estimation.

6.2.2 Model

We present INFINITY: Implicit Neural Fields for INterpretIng geomeTry and inferring phYsics.

Modulated INR In our model, we will treat each geometric input (d or n) or physical output function (v, p, or ν) separately and each will be modeled by an INR. Let us then consider a generic function u, which will represent either an input geometric field or an output physical field defined over a domain Ω or at its boundary $\partial\Omega$. Let us denote u_i the function corresponding to a specific airfoil example. u_i will be represented by an INR $f_{\theta_u,\phi_{u_i}}$ with two sets of parameters: parameters θ_u shared by all the u_i, and modulation parameters ϕ_{u_i} specific to each individual function u_i. In our airfoil example, ϕ_{u_i} enables the INR to handle different geometries. Overall, this decomposition allows the modulated INR to capture both shared characteristics among the example's functions u_i and the unique properties of each one. INFINITY leverages latent representations inferred from the modulation spaces of the INRs. These latent representations, denoted as z_{u_i}, are compact codes that encode information from the INRs' parameters. They serve as inputs to a hypernetwork h_u, with weights w_u, which computes the modulation parameters $\phi_{u_i} = h_u(z_{u_i})$.

In this work we use Fourier Features (Tancik et al., 2020b) as an INR backbone and apply shift modulation (Perez et al., 2018a): $f_{\theta,\phi_{u_i}}(\boldsymbol{x}) = \boldsymbol{W}_L\big(\chi_{L-1} \circ \chi_{L-2} \circ \cdots \circ \chi_0(\boldsymbol{x})\big) + \boldsymbol{b}_L$, with $\chi_j(\eta_j) = \sigma\big(\boldsymbol{W}_j\eta_j + \boldsymbol{b}_j + (\boldsymbol{\phi_{u_i}})_j\big)$. We note $\eta_0 = \boldsymbol{x}$ and $(\eta_j)_{j\geq 1}$ the hidden activations throughout the network. Hence, the parameters $\theta = (\boldsymbol{W}_j, \boldsymbol{b}_j)_{j=0}^{L}$ are shared between all examples and the modulation $\phi_{u_i} = ((\boldsymbol{\phi_{u_i}})_j)_{j=0}^{L-1}$ is specific to a single example. We compute the modulation parameters $\phi_{u_i} = ((\boldsymbol{\phi_{u_i}})_j)_{j=0}^{L-1}$ from z with a linear hypernetwork.

With the learned shared parameters (θ_u, w_u), the modulated INR enables two processes: decoding and encoding (see Figure 6.1). Decoding refers to mapping a given code z_{u_i} to the corresponding INR function $f_{\theta_u,\phi_{u_i}}$, where $\phi_{u_i} = h_u(z_{u_i})$, while encoding involves generating a code z_{u_i} given a function u_i, providing a compact representation of the function within the modulation space of the INR.

To obtain the compact code z_{u_i} for reconstructing the original field u_i using the INR, an inverse problem is solved through a procedure called *auto-decoding*. The objective is to compress the necessary information into z_{u_i} such that the reconstructed value $\tilde{u}_i(\boldsymbol{x}) = f_{\theta_u,\phi_{u_i}}(\boldsymbol{x})$ approximates the original value $u_i(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathcal{X}_i$. The approximate solution to this inverse problem is computed iteratively through a gradient descent optimization process:

$$
\begin{aligned}
&z_{u_i}^{(0)} = 0, \\
&z_{u_i}^{(k+1)} = z_{u_i}^{(k)} - \alpha \nabla_{z_{u_i}^{(k)}} \mathcal{L}_{\mu_i}(f_{\theta_u,\phi_{u_i}^{(k)}}, u_i), \\
&\text{with } \phi_{u_i}^{(k)} = h_u(z_{u_i}^{(k)}) \text{ for } 0 \leq k \leq K-1.
\end{aligned}
\tag{6.1}
$$

where α is the inner loop learning rate, K the number of inner steps, and $\mathcal{L}_{\mu_i}(u_i, \tilde{u}_i) = \mathbb{E}_{\boldsymbol{x}\sim\mu_i}[(u_i(\boldsymbol{x}) - \tilde{u}_i(\boldsymbol{x}))^2]$ where μ_i is a measure defined through the observation grid $\mathcal{X}_i$ $\mu_i(\cdot) = \sum_{x\in\mathcal{X}_i} \delta_x(\cdot)$, with $\delta_x(\cdot)$ the Dirac measure.

As indicated before, we treat each input and output function independently: there are two input functions denoted as (d, n) and four output functions denoted as (v_x, v_y, p, ν_t). Each $u_i \in \{d, n, v_x, v_y, p, \nu_t\}$ is represented by a modulated INR $f_{\theta_u,\phi_{u_i}}$, where u_i stands for a field specific to an airfoil example. INFINITY then learns a mapping between the latent representations of the geometric input fields and the latent representations of the physics output fields.

Inference As illustrated in Figure 6.1, INFINITY follows a three-step procedure: *encode*, *process*, and *decode*.

- *Encode:* Given the geometric input functions d_i, n_i and the corresponding INR learned parameters, respectively θ_d, w_d and θ_n, w_n, functions d_i, n_i are encoded into the latent codes z_{d_i}, z_{n_i} according to Equation (6.1). Since we can query the INRs anywhere within the domain, we can hence freely encode functions without mesh constraints. This lets us freely encode inputs with different geometries.

- *Process:* Once we obtain z_{d_i} and z_{n_i}, we can infer the latent output codes $\left(\hat{z}_{v_{xi}}, \hat{z}_{v_{y_i}}, \hat{z}_{p_i}, \hat{z}_{\nu_{ti}}\right) = g_\psi\Big(\left(z_{d_i}, z_{n_i}, V_{xi}, V_{y_i}\right)\Big)$. We consider here that g_ψ is implemented through an MLP with parameters ψ.

- *Decode:* We decode each processed output code $\left(\hat{z}_{v_{xi}}, \hat{z}_{v_{y_i}}, \hat{z}_{p_i}, \hat{z}_{\nu_{ti}}\right)$ with their associated hypernetwork and modulated INR. We make use of the INRs to freely query a physical field at any point within its spatial domain. These components generate the final output functions by mapping the latent codes back to the output space.

We present the inference pipeline in details in Algorithm 1.

6.2.3 Training

We implement a two-step training procedure that first learns the modulated INR parameters θ_u and ϕ_{u_i} for all input and output functions, before training the map g_ψ. During the training of the INRs we force the *auto-decoding* process to take only a few gradient steps to encode the geometric functions or physical fields. This enhances the INR capability to encode new geometrical inputs in a few steps at test time, and also reduces the space size of the target output codes. This regularization prevents the different INRs to memorize the training sets into the individual codes. In order to obtain a network that is capable of quickly encoding new geometrical and

Algorithm 1: Inference given d, n, V_x, V_y

Set codes to zero: $z_d \leftarrow 0$; $z_n \leftarrow 0$
Perform input encoding inner loop:
for step in $\{1, ..., K\}$ **do**
    $\phi_d = h_d(z_d)$; $\phi_n = h_n(z_n)$
    $z_d \leftarrow z_d - \alpha \nabla_{z_d} \mathcal{L}_{\mathcal{X}}(f_{\theta_d, \phi_d}, d)$
    $z_n \leftarrow z_n - \alpha \nabla_{z_n} \mathcal{L}_{\mathcal{X}}(f_{\theta_n, \phi_n}, n)$
end for
Process geometry and inlet velocities:
$\left(\hat{z}_{v_{x_i}}, \hat{z}_{v_{y_i}}, \hat{z}_{p_i}, \hat{z}_{\nu_{t_i}}\right) = g_\psi\left(\left(z_{d_i}, z_{n_i}, V_{xi}, V_{y_i}\right)\right)$
Query the predicted fields at location x*:*
$\hat{v}_x(x) = f_{\theta_{v_x}, \hat{\phi}_{v_x}}(x)$; $\hat{v}_y(x) = f_{\theta_{v_y}, \hat{\phi}_{v_y}}(x)$
$\hat{p}(x) = f_{\theta_p, \hat{\phi}_p}(x)$; $\hat{\nu}_t(x) = f_{\theta_\nu, \hat{\phi}_\nu}(x)$

physical inputs, we employ a second-order meta-learning training algorithm based on CAVIA (Zintgraf et al., 2019b). Compared to a first-order scheme such as Reptile (Nichol et al., 2018), the outer loop back-propagates the gradient through the K inner steps, consuming more memory. Indeed, we need to compute gradients of gradients but this yields higher reconstruction results with the modulated INR. We experimentally found that using 3 inner-steps for training, or testing, was sufficient to obtain very low reconstruction errors for the geometric or physical fields. Using more inner-steps would result in a higher computation cost with only a marginal gain in reconstruction capacity. We outline the training pipeline of a modulated INR in Algorithm 2.

Once the different INRs have been fitted, we encode the functions into the input codes z_{d_i}, z_{n_i} and target codes $z_{v_{x_i}}, z_{v_{y_i}}, z_{p_i}, z_{\nu_{t_i}}$. The training of g_ψ is performed in the small dimensional z-code space, and is supervised through the MSE loss with the target codes (see Algorithm 3 for details).

6.3 Experiments

Baselines We use the same baselines as Bonnet et al. (2022); GraphSAGE (Hamilton et al., 2017a), a PointNet (Qi et al., 2017), a Graph U-Net (Gao and Ji, 2019) and a MLP. Those baselines have been initially chosen as they process in different ways the inputs. The results are given for the setup "full data regime" of AirfRANS, using 800 samples for training and 200 for testing.

Results In Table 6.1, the INFINITY model demonstrates superior inference capabilities on the volume and surface compared to the baselines. Indeed, It achieves

Algorithm 2: Modulated INR training

while convergence is false **do**
  Sample a batch $\mathcal{B}$ *of data* $(u_i)_{i\in\mathcal{B}}$
  Set codes to zero: $z_{u_i} \leftarrow 0$ for i in $\mathcal{B}$
  Perform input encoding inner loop:
  for i in $\mathcal{B}$ and step in $\{1, ..., K_u\}$ **do**
    $\phi_{u_i} = h_u(z_{u_i})$
    $z_{u_i} \leftarrow z_{u_i} - \alpha_u \nabla_{z_{u_i}} \mathcal{L}_{\mathcal{X}_i}(f_{\theta_u,\phi_{u_i}}, u_i)$
  end for
  for i in $\mathcal{B}$: **do**
    $\phi_{u_i} = h_u(z_{u_i})$
  end for
  Perform outer loop update:
  $\theta_u \leftarrow \theta_u - \eta \frac{1}{|\mathcal{B}|} \sum_{i\in\mathcal{B}} \nabla_{\theta_u} \mathcal{L}_{\mathcal{X}_i}(f_{\theta_u,\phi_{u_i}}, u_i)$
  $w_u \leftarrow w_u - \eta \frac{1}{|\mathcal{B}|} \sum_{i\in\mathcal{B}} \nabla_{w_u} \mathcal{L}_{\mathcal{X}_i}(f_{\theta_u,\phi_{u_i}}, u_i)$
end while

Algorithm 3: Processor training

while convergence is false **do**
  Sample a batch $\mathcal{B}$ *of data:*
    $\left(z_{d_i}, z_{n_i}, V_{xi}, V_{y_i}, z_{v_{xi}}, z_{v_{y_i}}, z_{p_i}, z_{\nu_{ti}}\right)_{i\in\mathcal{B}}$
  Process geometry and inlet velocities:
  $\left(\hat{z}_{v_{xi}}, \hat{z}_{v_{y_i}}, \hat{z}_{p_i}, \hat{z}_{\nu_{ti}}\right) = g_\psi\Big(\left(z_{d_i}, z_{n_i}, V_{xi}, V_{y_i}\right)\Big)$
  Update parameters using the MSE Loss:
  $\text{MSE} = \frac{1}{|\mathcal{B}|} \sum_{i\in\mathcal{B}} \frac{1}{4}(||\hat{z}_{v_{xi}} - z_{v_{xi}}||^2 + ||\hat{z}_{v_{y_i}} - z_{v_{y_i}}||^2 + ||\hat{z}_{p_i} - z_{p_i}||^2 + ||\hat{z}_{\nu_{ti}} - z_{\nu_{ti}}||^2)$
end while

		INFINITY	GraphSAGE	MLP	Graph U-Net	PointNet
Volume	v_x	**0.06 ± 0.01**	0.83 ± 0.01	0.95 ± 0.06	1.52 ± 0.34	3.50 ± 1.04
	v_y	**0.06 ± 0.01**	0.99 ± 0.05	0.98 ± 0.17	2.03 ± 0.39	3.64 ± 1.26
	p	**0.25 ± 0.01**	0.66 ± 0.05	0.74 ± 0.13	0.66 ± 0.08	1.15 ± 0.23
	ν_t	**1.32 ± 0.08**	1.60 ± 0.21	1.90 ± 0.10	1.46 ± 0.14	2.92 ± 0.48
Surface	$p_{\|\mathcal{S}}$	**0.07 ± 0.01**	0.66 ± 0.10	1.13 ± 0.14	0.39 ± 0.07	0.93 ± 0.26
Relative error	C_D	**0.366 ± 0.023**	4.050 ± 0.704	4.289 ± 0.679	10.385 ± 1.895	14.637 ± 3.668
	C_L	**0.081 ± 0.007**	0.517 ± 0.162	0.767 ± 0.108	0.489 ± 0.105	0.742 ± 0.186
Spearman correlation	ρ_D	**0.578 ± 0.050**	-0.303 ± 0.124	-0.117 ± 0.256	-0.138 ± 0.258	-0.022 ± 0.097
	ρ_L	**0.997 ± 0.001**	0.965 ± 0.011	0.913 ± 0.018	0.967 ± 0.019	0.938 ± 0.023
Inference time (μs)		98 ± 70	20.9 ± 2.3	13.3 ± 0.2	357.8 ± 36.9	33.9 ± 3.5

Table 6.1: Test results on AirfRANS. Mean squared error (MSE) on normalized fields expressed with factor ($\times 10^{-2}$) for the volume and ($\times 10^{-1}$) for the surface. Relative errors C_D, C_L on the drag and lift and Spearman correlations ρ_D, ρ_L on the drag and lift. The results from the baselines are taken from Bonnet et al. (2022).

significantly lower error values on the volume velocity and pressure fields, while exhibiting an order-of-magnitude lower MSE on the surface pressure. This substantial gain in prediction power translates to order of magnitude lower relative errors on the drag and lift forces, accompanied by high positive Spearman correlations. These results indicate a strong alignment between INFINITY's predictions and the true drag and lift forces. Consequently, INFINITY emerges as the only model capable of predicting accurately physical fields on the volume and surface while maintaining coherent and accurate drag and lift estimations. On the downside, the INFINITY model has a longer inference time compared to GraphSAGE and PointNet. However, this increased inference time is still within an acceptable range, considering its superior performance and that a numerical solver needs approximately 20 minutes to complete a simulation. Furthermore, it is counterbalanced by the ability to query the full mesh directly, in stark contrast to graph-based methods that necessitate sub-sampling to process the inputs.

6.4 Conclusion

We introduce INFINITY, a model that utilizes coordinate-based networks to encode geometric information and physical fields into compact representations. INFINITY establishes a mapping between geometry and physical fields within a reduced representation space. We validated our model on AirfRANS, a challenging dataset for the Reynolds-Averaged Navier-Stokes equation, where it significantly outperforms previous baselines across all relevant performance metrics. At post-processing stage, the predicted fields yield accurate lift and drag forces. This validates INFINITY's potential as a surrogate design model, where it could be plugged in any design optimization or exploration loop.

Chapter 7

Preserving Spatial Structure for Latent PDE Modeling with Local Neural Fields

We present AROMA (Attentive Reduced Order Model with Attention), a framework designed to enhance the modeling of partial differential equations (PDEs) using local neural fields. Our flexible encoder-decoder architecture can obtain smooth latent representations of spatial physical fields from a variety of data types, including irregular-grid inputs and point clouds. This versatility eliminates the need for patching and allows efficient processing of diverse geometries. The sequential nature of our latent representation can be interpreted spatially and permits the use of a conditional transformer for modeling the temporal dynamics of PDEs. By employing a diffusion-based formulation, we achieve greater stability and enable longer rollouts compared to conventional MSE training. AROMA's superior performance in simulating 1D and 2D equations underscores the efficacy of our approach in capturing complex dynamical behaviors. *Github page*: https://github.com/LouisSerrano/aroma

Serrano, L., Wang, T. X., Naour, E. L., Vittaut, J. N., Gallinari, P. (2024). AROMA: Preserving Spatial Structure for Latent PDE Modeling with Local Neural Fields. Neurips 2024.

7.1 Introduction

In recent years, many deep learning (DL) surrogate models have been introduced to approximate solutions to partial differential equations (PDEs) (Lu et al., 2021a; Li et al., 2021; Brandstetter et al., 2022d; Stachenfeld et al., 2022). Among these, the family of neural operators has been extensively adopted and tested across various scientific domains, demonstrating the potential of data-centric DL models in science (Pathak et al., 2022; Vinuesa and Brunton, 2022).

Neural Operators were initially constrained by discretization and domain geometry limitations. Recent advancements, such as neural fields (Yin et al., 2022b; Serrano et al., 2023) and transformer architectures (Li et al., 2023a; Hao et al., 2023), have partially addressed these issues, improving both dynamic modeling and steady-state settings. However, Neural Fields struggle to model spatial information and local dynamics effectively, and existing transformer architectures, while being flexible, are computationally expensive due to their operation in the original physical space and require large training datasets.

Our hypothesis is that considering spatiality is essential in modeling spatio-temporal phenomena, yet applying attention mechanisms directly is computationally expensive. We propose a new framework that models the dynamics in a reduced latent space, encoding spatial information compactly, by one or two orders of magnitude relative to the original space. This approach addresses both the complexity issues of transformer architectures and the spatiality challenges of Neural Fields.

Our novel framework leverages attention blocks and neural fields, resulting in a model that is easy to train and achieves state-of-the-art results on most datasets, particularly for complex geometries, without requiring prior feature engineering. To the best of our knowledge, we are the first to propose a fully attention-based architecture for processing domain geometries and unrolling dynamics. Compared to existing transformer architectures for PDEs, our framework first encapsulates the domain geometry and observation values in a compact latent representation, efficiently forecasting the dynamics at a lower computational cost. Transformer-based methods such as (Li et al., 2023a; Hao et al., 2023) unroll the dynamics in the original space, leading to high complexity.

Our contributions are summarized as follows:

- A principled and versatile encode-process-decode framework for solving PDEs that operate on general input geometries, including point sets, grids, or meshes, and can be queried at any location within the spatial domain.
- A new spatial encode / process / decode approach: Variable-size inputs are mapped onto a fixed-size compact latent token space that encodes local spatial information. This latent representation is further processed by a transformer architecture that models the dynamics while exploiting spatial relations both at the local token level and globally across tokens. The decoding exploits a conditional neural field, allowing us to query forecast values at any point in the spatial domain of the equation.
- We include stochastic components at the encoding and processing levels to enhance stability and forecasting accuracy.
- Experiments performed on representative spatio-temporal forecasting problems demonstrate that AROMA is on par with or outperforms state-of-the-art baselines in terms of both accuracy and complexity.

7.2 Problem setting

In this paper, we focus on time-dependent PDEs defined over a spatial domain Ω (with boundary $\partial\Omega$) and temporal domain $[0, T]$. In the general form, their solutions $\boldsymbol{u}(x, t)$ satisfy the following constraints :

$$\frac{\partial \boldsymbol{u}}{\partial t} = F\left(\nu, t, x, \boldsymbol{u}, \frac{\partial \boldsymbol{u}}{\partial x}, \frac{\partial^2 \boldsymbol{u}}{\partial x^2}, \dots\right), \quad \forall x \in \Omega, \forall t \in (0, T] \tag{7.1}$$

$$\mathcal{B}(\boldsymbol{u})(t, x) = 0 \quad \forall x \in \partial\Omega, \forall t \in (0, T] \tag{7.2}$$

$$\boldsymbol{u}(0, x) = \boldsymbol{u}^0 \quad \forall x \in \Omega \tag{7.3}$$

where ν represents a set of PDE coefficients, Equations (2) and (3) represent the constraints with respect to the boundary and initial conditions. We aim to learn, using solutions data obtained with classical solvers, the evolution operator $\mathcal{G}$ that predicts the state of the system at the next time step: $\boldsymbol{u}^{t+\Delta t} = \mathcal{G}(\boldsymbol{u}^t)$. We have access to training trajectories obtained with different initial conditions, and we want to generate accurate trajectory rollouts for new initial conditions at test time. A rollout is obtained by the iterative application of the evolution operator $\boldsymbol{u}^{m\Delta t} = \mathcal{G}^m(\boldsymbol{u}^0)$.

7.3 Model Description

7.3.1 Model overview

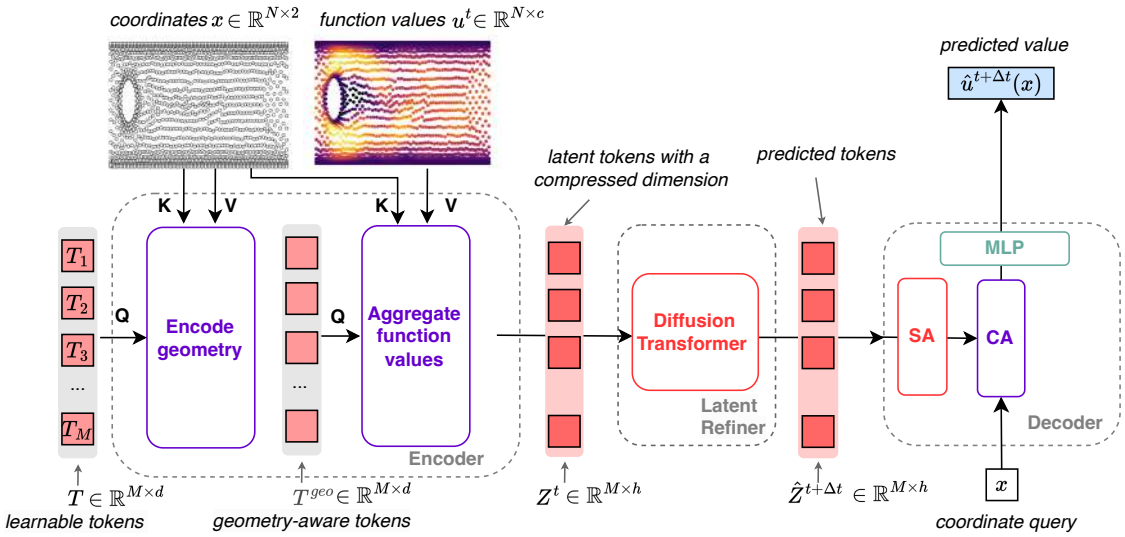


Figure 7.1: AROMA inference: The discretization-free encoder compresses the information of a set of N input values to a sequence of M latent tokens, where $M < N$. The conditional diffusion transformer is used to model the dynamics, acting as a latent refiner. The continuous decoder leverages self-attentions (SA), cross-attention (CA) and a local INR to map back to the physical space. Learnable tokens are shared and encode spatial relations. Latent token Z^t represents u_t and $Z^{t+\Delta t}$ is the prediction corresponding to $u_{t+\Delta t}$.

We provide below an overview of the global framework and each component is described in a subsequent section. The model comprises three key components, as detailed in Figure 7.1.

- **Encoder** $\mathcal{E}_w : \boldsymbol{u}^t_{\mathcal{X}} \to \boldsymbol{Z}^t$. The encoder takes input values $\boldsymbol{u}^t_{\mathcal{X}}$ sampled over the domain Ω at time t, where $\mathcal{X}$ denotes the discrete sample space and could be a grid, an irregular mesh or a point set. $\boldsymbol{u}^t_{\mathcal{X}}$ is observed at locations $\boldsymbol{x} = (x_1, \dots x_N)$, with values $\boldsymbol{u}^t = (\boldsymbol{u}^t(\boldsymbol{x}_1), \cdots, \boldsymbol{u}^t(\boldsymbol{x}_N))$. N is the number of observations and can vary across samples. $\boldsymbol{u}^t_{\mathcal{X}}$ is projected through a cross attention mechanism onto a set of M tokens $\boldsymbol{Z}^t = (\boldsymbol{z}^t_1, \cdots, \boldsymbol{z}^t_M)$ with M a fixed parameter. This allows mapping any discretized input $\boldsymbol{u}^t_{\mathcal{X}}$ onto a fixed dimensional latent representation $\boldsymbol{Z}^t$ encoding implicit local spatial information from the input domain. The encoder is trained as a VAE and $\boldsymbol{Z}^t$ is sampled from a multivariate normal statistics as detailed in Section 7.3.2.
- **Latent time-marching refiner** $\mathcal{R}_\theta : \boldsymbol{Z}^t \to \hat{\boldsymbol{Z}}^{t+\Delta t}$. We model the dynamics in the latent space through a transformer. The dynamics can be unrolled auto-

regressively in the latent space for any time horizon without requiring to project back in the original domain Ω. Self-attention operates on the latent tokens, which allows modeling global spatial relations between the local token representations. The transformer is enriched with a conditional diffusion mechanism operating between two successive time steps of the transformer. We experimentally observed that this probabilistic model was more robust than a baseline deterministic transformer for temporal extrapolation.

- **Decoder** $\mathcal{D}_\psi : \hat{\boldsymbol{Z}}^{t+\Delta t} \to \hat{\boldsymbol{u}}^{t+\Delta t}$. The decoder uses the latent tokens $\hat{\boldsymbol{Z}}^{t+\Delta t}$ to approximate the function value $\hat{\boldsymbol{u}}^{t+\Delta t}(x) = \mathcal{D}_\psi(x, \hat{\boldsymbol{Z}}^{t+\Delta t})$ for any query coordinate $x \in \Omega$. We therefore denote $\hat{\boldsymbol{u}}^{t+\Delta t} = \mathcal{D}_\psi(\boldsymbol{Z}^{t+\Delta t})$ the predicted function.

Inference We encode the initial condition and unroll the dynamics in the latent space by successive denoisings: $\hat{\boldsymbol{u}}^{m\Delta t} = \mathcal{D}_\psi \circ \mathcal{R}_\theta^m \circ \mathcal{E}_w(\boldsymbol{u}^0)$. We then decode along the trajectory to get the reconstructions. We outline the full inference pipeline in Figure 7.1 and detail its complexity analysis in Appendix B.3.1.

Training We perform a two-stage training: we first train the encoder and decoder, secondly train the refiner. This is more stable than end-to-end training.

7.3.2 Encoder-decoder description

The encoder-decoder components are jointly trained using a VAE setting. The encoder is specifically designed to capture local input observation from any sampled point set in the spatial domain and encodes this information into a fixed number of tokens. The decoder can be queried at any position in the spatial domain, irrespective of the input sample.

Encoder The encoder maps an arbitrary number N of observations $(\boldsymbol{x}, \boldsymbol{u}(\boldsymbol{x})) := ((x_1, \boldsymbol{u}(x_1)), \dots, (x_N, \boldsymbol{u}(x_N))$ onto a latent representation $\boldsymbol{Z}$ of fixed size M through the following series of transformations:

$$
\begin{aligned}
&\text{(i)} && (\boldsymbol{x}, \boldsymbol{u}(\boldsymbol{x})) && \xrightarrow{\text{(positional, value) embeddings}} && (\boldsymbol{\gamma}(\boldsymbol{x}), \boldsymbol{v}(\boldsymbol{x})) \in \mathbb{R}^{N\times d} \\
&\text{(ii)} && (\mathbf{T}, \boldsymbol{\gamma}(\boldsymbol{x})) && \xrightarrow{\text{geometry encoding}} && \mathbf{T}^{\text{geo}} \in \mathbb{R}^{M\times d} \\
&\text{(iii)} && (\mathbf{T}^{\text{geo}}, \boldsymbol{v}(\boldsymbol{x})) && \xrightarrow{\text{observation spatial encoding}} && \mathbf{T}^{\text{obs}} \in \mathbb{R}^{M\times d} \\
&\text{(iv)} && \mathbf{T}^{\text{obs}} && \xrightarrow{\text{dimension reduction}} && \boldsymbol{Z} \in \mathbb{R}^{M\times h}
\end{aligned}
$$

where $(\boldsymbol{\gamma}(\boldsymbol{x}), \boldsymbol{v}(\boldsymbol{x})) = ((\gamma(x_1), v(x_1)), \dots, (\gamma(x_N), v(x_N)))$, and $h \ll d$.

(i) Embed positions and observations: Given an input sequence of coordinate-value pairs $(x_1, \boldsymbol{u}(x_1)), \ldots, (x_N, \boldsymbol{u}(x_N))$, we construct sequences of positional embeddings $\boldsymbol{\gamma} = (\gamma(x_1), \ldots, \gamma(x_N))$ and value embeddings $\boldsymbol{v} = (v(x_1), \ldots, v(x_N))$, where $\gamma(x) = \texttt{FourierFeatures}(x; \omega)$ and $v(x) = \texttt{Linear}(\boldsymbol{u}(x))$, with ω a fixed set of frequencies. These embeddings are aggregated onto a smaller set of learnable query tokens $\mathbf{T} = (T_1, \ldots, T_M)$ and then $\mathbf{T}' = (T'_1, \ldots, T'_M)$ with M fixed, to compress the information and encode the geometry and spatial latent representations.

(ii) Encode geometry: Geometry-aware tokens $\mathbf{T}$ are obtained with a multi-head cross-attention layer and a feedforward network (FFN), expressed as $\mathbf{T}^{\text{geo}} = \mathbf{T} + \texttt{FFN}(\texttt{CrossAttention}(\mathbf{Q} = \mathbf{W}_Q\mathbf{T}, \mathbf{K} = \mathbf{W}_K\boldsymbol{\gamma}, \mathbf{V} = \mathbf{W}_V\boldsymbol{\gamma}))$. This step does not include information on the observations, ensuring that similar geometries yield similar query tokens $\mathbf{T}^{\text{geo}}$ irrespective of the $\boldsymbol{u}$ values.

(iii) Encode observations: The $\mathbf{T}^{\text{geo}}$ tokens are then used to aggregate the observation values via a cross-attention mechanism: $\mathbf{T}^{\text{obs}} = \mathbf{T}^{\text{geo}} + \texttt{FFN}(\texttt{CrossAttention}(\mathbf{Q} = \mathbf{W}'_Q\mathbf{T}^{\text{geo}}, \mathbf{K} = \mathbf{W}'_K\boldsymbol{\gamma}, \mathbf{V} = \mathbf{W}'_V\boldsymbol{v}))$. Here, the values contain information on the observation values, and the keys contain information on the observation locations.

(iv) Reduce channel dimension and sample $\boldsymbol{Z}$: The information in the channel dimension of $\mathbf{T}'$ is compressed using a bottleneck linear layer. To avoid exploding variance in this compressed latent space, we regularize it with a penalty on the L_2 norm of the latent code $\|\boldsymbol{Z}\|^2$. Introducing stochasticity through a variational formulation further helps to regularize the auto-encoding and obtain smoother representations for the forecasting step. For this, we learn the components of a Gaussian multivariate distribution $\boldsymbol{\mu} = \texttt{Linear}(\mathbf{T}^{\text{obs}})$ and $\log(\boldsymbol{\sigma}) = \texttt{Linear}(\mathbf{T}^{\text{obs}})$ from which the final token embedding $\boldsymbol{Z}$ is sampled.

Decoder The decoder's role is to reconstruct $\hat{\boldsymbol{u}}^{t+\Delta t}$ from $\hat{\boldsymbol{Z}}^{t+\Delta t}$, see Figure 7.1. Since training is performed in two steps ("encode-decode" first and then "process"), the decoder is trained to reconstruct $\hat{\boldsymbol{u}}^t$ for input $\boldsymbol{u}^t$. One proceeds as follows. **(i) Increase channel dimensions and apply self-attention**: The decoder first lifts the latent tokens $\boldsymbol{Z}$ to a higher channel dimension (this is the reverse operation of the one performed by the encoder) and then apply several layers of self-attention to get tokens $\boldsymbol{Z}'$. **(ii) Cross-attend**: The decoder applies cross-attention to obtain feature vectors that depend on the query coordinate x, $(\mathbf{f}^u_q(x)) = \texttt{CrossAttention}(\mathbf{Q} = \mathbf{W}_Q(\gamma_q(x)), \mathbf{K} = \mathbf{W}_K\boldsymbol{Z}', \mathbf{V} = \mathbf{W}_V\boldsymbol{Z}')$, where γ_q is a Fourier features embedding of bandwidth ω_q. **(iii) Decode with MLP**: Finally, we use a small MLP to decode this feature vector and obtain the reconstruction $\hat{\boldsymbol{u}}(x) = \text{MLP}(\mathbf{f}^u_q(x))$. In contrast with existing neural field methods for dynamics modeling, the feature vector here is local. In practice, one uses multiple cross attentions to get feature vectors with different frequencies (see Appendix Figures B.4 and B.5 for further details).

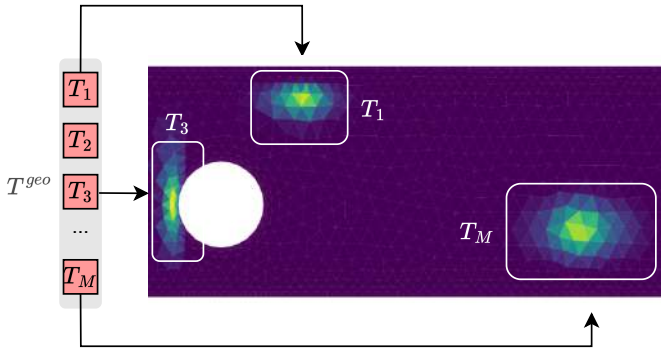


Figure 7.2: Spatial interpretation of the tokens through cross attention between T^{geo} and $\boldsymbol{\gamma}(x)$ for each x in the domain. Here we visualize the cross-attention of three different tokens for a given head. The cross attentions can have varying receptive fields depending on the geometries.

Training The encoder and decoder are jointly optimized as a variational autoencoder (VAE) (Kingma and Welling, 2013) to minimize the following objective : $\mathcal{L} = \mathcal{L}_{\text{recon}} + \beta \cdot \mathcal{L}_{KL}$; where $\mathcal{L}_{\text{recon}} = \text{MSE}(u^t_{\mathcal{X}}, \hat{u}^t_{\mathcal{X}})$ is the reconstruction loss between the input and the reconstruction $\mathcal{D}_\psi(\boldsymbol{Z}^t, \mathcal{X})$ on the grid $\mathcal{X}$, with $\boldsymbol{Z}^t \sim \mathcal{N}(\boldsymbol{\mu}^t, (\boldsymbol{\sigma}^t)^2)$ and $\boldsymbol{\mu}^t, \boldsymbol{\sigma}^t = \mathcal{E}_w(u^t_{\mathcal{X}})$. The KL divergence loss $\mathcal{L}_{\text{KL}} = D_{\text{KL}}(\mathcal{N}(\boldsymbol{\mu}^t, (\boldsymbol{\sigma}^t)^2) \,||\, \mathcal{N}(0, I))$ helps regularize the network and prevents overfitting. We found that using a variational formulation was essential to obtain smooth latent representations while training the encoder-decoder.

7.3.3 Transformer-based diffusion

Modeling the dynamics is performed in the latent $\boldsymbol{Z}$ space. This space encodes spatial information present in the original space while being a condensed, smaller-sized representation, allowing for reduced complexity dynamics modeling. As indicated, the dynamics can be unrolled auto-regressively in this space for any time horizon without the need to map back to the original space. We use absolute positional embeddings E_{pos} and a linear layer to project onto a higher dimensional space: $\boldsymbol{Z}_{[0]} = \texttt{Linear}(\boldsymbol{Z}) + E_{\text{pos}}$. The backbone then applies several self-attention blocks, which process tokens as follows:

$$\boldsymbol{Z}_{[l+1]} \leftarrow \boldsymbol{Z}_{[l]} + \texttt{Attention}(\texttt{LayerNorm}(\boldsymbol{Z}_{[l]})) \tag{7.4}$$

$$\boldsymbol{Z}_{[l+1]} \leftarrow \boldsymbol{Z}_{[l+1]} + \texttt{FFN}(\texttt{LayerNorm}(\boldsymbol{Z}_{[l+1]}) \tag{7.5}$$

We found out that adding a diffusion component to the transformer helped enhance the stability and allowed longer forecasts. Diffusion steps are inserted between two time steps t and $t + \Delta t$ of the time-marching process transformer. The diffusion steps are denoted by k and are different from the ones of the time-marching

process (several diffusion steps k are performed between two time-marching steps t and $t + \Delta t$).
We then use a conditional diffusion transformer architecture close to Peebles and Xie (2023) for $\mathcal{R}_\theta$, where we detail the main block in Appendix B.2. At diffusion step k, the input to the network is a sequence stacking the tokens at time t and the current noisy targets estimate $(\boldsymbol{Z}^t, \tilde{\boldsymbol{Z}}_k^{t+\Delta t})$. See Appendix B.2, Figure B.1 and Figure B.2 for more details. To train the diffusion transformer $\mathcal{R}_\theta$, we freeze the encoder and decoder, and use the encoder to sample pairs of successive latent tokens $(\boldsymbol{Z}^t, \boldsymbol{Z}^{t+\Delta t})$. We employ the "v-predict" formulation of DDPM (Salimans and Ho, 2022) for training and sampling.

7.4 Experiments

In this section, we systematically evaluate the performance of our proposed model across various experimental settings, focusing on its ability to handle dynamics on both regular and irregular grids. First, we investigate the dynamics on regular grids, where we benchmark our model against state-of-the-art neural operators, including Fourier Neural Operators (FNO), ResNet, Neural Fields, and Transformers. This comparison highlights the efficacy of our approach in capturing complex spatio-temporal patterns on structured domains. Second, we extend our analysis to dynamics on irregular grids and shared geometries, emphasizing the model's extrapolation capabilities in data-constrained regimes. Here, we compare our results with Neural Fields and Transformers, demonstrating the robustness of our model in handling less structured and more complex spatial configurations. Lastly, we assess the model's capacity to process diverse geometries and underlying spatial representations by comparing its performance on irregular grids and different geometries. This evaluation highlights the flexibility and generalization ability of our model in encoding and learning from varied spatial domains, showcasing its potential in accurately representing and predicting dynamics across a wide range of geometric settings. We include additional results from ablation studies in Appendix B.3.6.

7.4.1 Dynamics on regular grids

We begin our analysis with dynamics modeling on regular grid settings. Though our model is targeted for complex geometries, we believe this scenario remains an important benchmark to assess the efficiency of surrogate models.

Datasets • **1D Burgers' Equation** (*Burgers*): Models shock waves, using a dataset with periodic initial conditions and forcing term as in Brandstetter et al. (2022d). It includes 2048 training and 128 test trajectories, at resolutions of

Table 7.1: Model Performance Comparison - Test results. Metrics in Relative L_2.

Model	*Burgers*	*Navier-Stokes* 1×10^{-4}	*Navier-Stokes* 1×10^{-5}
FNO	5.00×10^{-2}	$\underline{1.53 \times 10^{-1}}$	$\mathbf{1.24 \times 10^{-1}}$
ResNet	8.50×10^{-2}	3.77×10^{-1}	2.56×10^{-1}
DINO	4.57×10^{-1}	7.25×10^{-1}	3.72×10^{-1}
CORAL	6.20×10^{-2}	3.77×10^{-1}	3.11×10^{-1}
GNOT	1.28×10^{-1}	1.85×10^{-1}	1.65×10^{-1}
OFormer	$\underline{4.92 \times 10^{-2}}$	1.36×10^{-1}	2.40×10^{-1}
AROMA	$\mathbf{3.65 \times 10^{-2}}$	$\mathbf{1.05 \times 10^{-1}}$	$\mathbf{1.24 \times 10^{-1}}$

$(250, 100)$. We create sub-trajectories of 50 timestamps and treat them independently. • **2D Navier Stokes Equation**: for a viscous and incompressible fluid. We use the data from Li et al. (2021). The equation is expressed with the vorticity form on the unit torus: $\frac{\partial w}{\partial t} + u \cdot \nabla w = \nu \Delta w + f$, $\nabla u = 0$ for $x \in \Omega, t > 0$, where ν is the viscosity coefficient. We consider two different versions $\nu = 10^{-4}$ (*Navier-Stokes* 1×10^{-4}) and $\nu = 10^{-5}$ (*Navier-Stokes* 1×10^{-5}), and use train and test sets of 1000 and 200 trajectories with a base spatial resolution of size 64×64. We consider a horizon of $T = 30$ for $\nu = 10^{-4}$ and $T = 20$ for $\nu = 10^{-5}$ since the phenomenon is more turbulent. At test time, we use the vorticity at $t_0 = 10$ as the initial condition.

Setting We train all the models with supervision on the next state prediction to learn to approximate the time-stepping operator $\boldsymbol{u}^{t+\Delta t} = \mathcal{G}(\boldsymbol{u}^t)$. At test time, we unroll the dynamics auto-regressively with each model and evaluate the prediction with a relative L_2 error defined as $L_2^{\text{test}} = \frac{1}{N_{\text{test}}} \sum_{j \in \text{test}} \frac{||\hat{u}_j^{\text{trajectory}} - u_j^{\text{trajectory}}||_2}{||u_j^{\text{trajectory}}||_2}$.

Baselines We use a diverse panel of baselines including state of the art regular-grid methods such as FNO (Li et al., 2021) and ResNet (He et al., 2016b; Lippe et al., 2023), flexible transformer architectures such as OFormer (Li et al., 2023a), and GNOT (Hao et al., 2023), and finally neural-field based methods with DINO (Yin et al., 2022b) and CORAL (Serrano et al., 2023).

Results Table 7.1 presents a comparison of model performance on the *Burgers*, *Navier-Stokes1e-4*, and *Navier-Stokes1e-5* datasets, with metrics reported in Relative L_2. Our method, AROMA, demonstrates excellent performance across the board, highlighting its ability to capture the dynamics of turbulent phenomena, as reflected in the *Navier-Stokes* datasets.

In contrast, DINO and CORAL, both global neural field models, perform poorly in capturing turbulent phenomena, exhibiting significantly higher errors compared to other models. This indicates their limitations in handling complex fluid dynamics.

On the other hand, AROMA outperforms GNOT on all datasets, though it performs reasonably well compared to the neural field based method.

Regarding the regular-grid methods, ResNet shows suboptimal performance in the pure teacher forcing setting, rapidly accumulating errors over time during inference. FNO stands out as the best baseline, demonstrating competitive performance on all datasets. We hypothesize that FNO's robustness to error accumulation during the rollout can be attributed to its Fourier block, which effectively cuts off high-frequency components. Overall, the results underscore AROMA's effectiveness and highlight the challenges Neural Field-based models face in accurately modeling complex phenomena.

7.4.2 Dynamics on irregular grids with shared geometries

We continue our experimental analysis with dynamics on unstructured grids, where we observe trajectories only through sparse spatial observations over time. We adopt a data-constrained regime and show that our model can still be competitive with existing Neural Fields in this scenario.

Datasets To evaluate our framework, we utilize two fluid dynamics datasets commonly used as a benchmark for this task (Yin et al., 2022b; Serrano et al., 2023) with unique initial conditions for each trajectory: • **2D Navier-Stokes Equation** (*Navier-Stokes* 1×10^{-3}): We use the same equation as in Section 7.4.1 but with a higher viscosity coefficient $\nu = 1e-3$. We have 256 trajectories of size 40 for training and 32 for testing. We used a standard resolution of 64x64. • **3D Shallow-Water Equation** (*Shallow-Water*): This equation approximates fluid flow on the Earth's surface. The data includes the vorticity w and height h of the fluid. The training set comprises 64 trajectories of size 40, and the test set comprises 8 trajectories with 40 timestamps. We use a standard spatial resolution of 64×128.

Setting • **Temporal Extrapolation**: For both datasets, we split trajectories into two equal parts of 20 timestamps each. The first half is denoted as *In-t* and the second half as *Out-t*. The training set consists of *In-t*. During training, we supervise with the next state only. During testing, the model unrolls the dynamics from a new initial condition (IC) up to the end of *Out-t*, i.e. for 39 steps. Evaluation within the *In-t* horizon assesses the model's ability to forecast within the training regime. The *Out-t* evaluation tests the model's extrapolation capabilities beyond the training horizon. • **Sparse observations**: For the train and test set we randomly select π percent of the available regular mesh to create a unique grid for each trajectory, both in the train and in the test. The grid is kept fixed along a given trajectory. While each grid is different, they maintain the same level of sparsity across trajectories. In our case, $\pi = 100\%$ amounts to the fully observable case, while in $\pi = 25\%$ each

grid contains around 1020 points for *Navier-Stokes* 1×10^{-3} and 2040 points for *Shallow-Water*.

Baselines We compare our model to OFormer (Li et al., 2023a), GNOT (Hao et al., 2023), and choose DINO (Yin et al., 2022b) and CORAL (Serrano et al., 2023) as the neural field baselines.

Training and evaluation During training, we only use the data from the training horizon (*In-t*). At test time, we evaluate the models to unroll the dynamics for new initial conditions in the training horizon (*In-t*) and for temporal extrapolation (*Out-t*).

Results Table 7.2 demonstrates that AROMA consistently achieves low MSE across all levels of observation sparsity and evaluation horizons for both datasets. Overall, our method performs best with some exceptions. On *Shallow-Water* our model is slightly outperformed by CORAL in the fully observed regime, potentially because of a lack of data. Similarly, on *Navier-Stokes* 1×10^{-3} CORAL has slightly better scores in the very sparse regime $\pi = 5\%$. Overall, this is not surprising as meta-learning models excel in data-constrained regimes. We believe our geometry-encoding block is crucial for obtaining good representations of the observed values in the sparse regimes, potentially explaining the performance gap with GNOT and OFormer.

Table 7.2: Temporal Extrapolation - Test results. Metrics in MSE.

$\mathcal{X}_{tr} \downarrow \mathcal{X}_{te}$	dataset →	*Navier-Stokes* 1×10^{-3}		*Shallow-Water*	
		In-t	*Out-t*	*In-t*	*Out-t*
$\pi = 100\%$	DINO	2.51×10^{-2}	9.91×10^{-2}	4.15×10^{-4}	3.55×10^{-3}
	CORAL	5.76×10^{-4}	3.00×10^{-3}	$\mathbf{2.12 \times 10^{-5}}$	$\mathbf{6.00 \times 10^{-4}}$
	OFormer	7.76×10^{-3}	6.39×10^{-2}	1.00×10^{-2}	2.23×10^{-2}
	GNOT	3.21×10^{-4}	2.33×10^{-3}	2.48×10^{-4}	2.17×10^{-3}
	AROMA	$\mathbf{1.32 \times 10^{-4}}$	$\mathbf{2.23 \times 10^{-3}}$	3.10×10^{-5}	8.75×10^{-4}
$\pi = 25\%$ irregular grid	DINO	3.27×10^{-2}	1.40×10^{-1}	4.12×10^{-4}	3.26×10^{-3}
	CORAL	1.54×10^{-3}	1.07×10^{-2}	3.77×10^{-4}	1.44×10^{-3}
	OFormer	3.73×10^{-2}	1.60×10^{-1}	6.19×10^{-3}	1.40×10^{-2}
	GNOT	2.07×10^{-2}	6.24×10^{-2}	8.91×10^{-4}	4.66×10^{-3}
	AROMA	$\mathbf{7.02 \times 10^{-4}}$	$\mathbf{6.31 \times 10^{-3}}$	$\mathbf{1.49 \times 10^{-4}}$	$\mathbf{1.02 \times 10^{-3}}$
$\pi = 5\%$ irregular grid	DINO	3.63×10^{-2}	1.35×10^{-1}	4.47×10^{-3}	9.88×10^{-3}
	CORAL	$\mathbf{2.87 \times 10^{-3}}$	$\mathbf{1.48 \times 10^{-2}}$	2.72×10^{-3}	6.58×10^{-3}
	OFormer	3.23×10^{-2}	1.12×10^{-1}	8.67×10^{-3}	1.72×10^{-2}
	GNOT	7.43×10^{-2}	1.89×10^{-1}	5.05×10^{-3}	1.49×10^{-2}
	AROMA	4.73×10^{-3}	2.01×10^{-2}	$\mathbf{1.93 \times 10^{-3}}$	$\mathbf{3.14 \times 10^{-3}}$

7.4.3 Dynamics on different geometries

Finally, we extend our analysis to learning dynamics over varying geometries.

Datasets We evaluate our model on two problems involving non-convex domains, as described by Pfaff et al. (2021). Both scenarios involve fluid dynamics in a domain with an obstacle, where the area near the boundary conditions (BC) is more finely discretized. The boundary conditions are specified by the mesh, and the models are trained with various obstacles and tested on different, yet similar, obstacles. • **Cylinder** (*CylinderFlow*): This dataset simulates water flow around a cylinder using a fixed 2D Eulerian mesh, representing *incompressible* fluids. For each node j in the mesh $\mathcal{X}$, we have data on the node position $x^{(j)}$, momentum $w(x^{(j)})$, and pressure $p(x^{(j)})$. Our task is to learn the mapping from $(w_t(x), p_t(x))_{x\in\mathcal{X}}$ to $(w_{t+\Delta t}(x), p_{t+\Delta t}(x))_{x\in\mathcal{X}}$ for a fixed Δt. • **Airfoil** (*AirfoilFlow*): This dataset simulates the aerodynamics around an airfoil, relevant for *compressible* fluids. In addition to the data available in the Cylinder dataset, we also have the fluid density $\rho(x^{(j)})$ for each node j. Our goal is to learn the mapping from $(w_t(x), p_t(x), \rho_t(x))_{x\in\mathcal{X}}$ to $(w_{t+\Delta t}(x), p_{t+\Delta t}(x), \rho_{t+\Delta t}(x))_{x\in\mathcal{X}}$. Each example in the dataset corresponds to a unique mesh. On average, there are 5233 nodes per mesh for *AirfoilFlow* and 1885 for *CylinderFlow*. We temporally subsample the original trajectories by taking one timestamp out of 10, forming trajectories of 60 timestamps. We use the first 40 timestamps for training (*In-t*) and keep the last 20 timestamps for evaluation (*Out-t*).

Setting We train all the models with supervision on the next state prediction. At test time, we unroll the dynamics auto-regressively with each model and evaluate the prediction with a mean squared error (MSE) both in the training horizon *(In-t)* and beyond the training horizon *(Out-t)*.

Results The results in Table 7.3 show that AROMA outperforms other models in predicting flow dynamics on both *CylinderFlow* and *AirfoilFlow* geometries, achieving the lowest MSE values across all tests. This indicates AROMA's superior ability to encode geometric features accurately. Additionally, AROMA maintains stability over extended prediction horizons, as evidenced by its consistently low *Out-t* MSE values.

7.4.4 Long rollouts and uncertainty quantification

After training different models on *Burgers*, we compare them on long trajectory rollouts. We start from $t_0 = 50$ (i.e. use a numerical solver for 50 steps), and unroll our dynamics auto-regressively for 200 steps. Note that all the models were

Table 7.3: Dynamics on different geometries - Test results. MSE on normalized data.

Model	*CylinderFlow*		*AirfoilFlow*	
	In-t	*Out-t*	*In-t*	*Out-t*
CORAL	$\underline{4.458 \times 10^{-2}}$	$\underline{8.695 \times 10^{-2}}$	$\underline{1.690 \times 10^{-1}}$	$\underline{3.420 \times 10^{-1}}$
DINO	1.349×10^{-1}	1.576×10^{-1}	3.770×10^{-1}	4.740×10^{-1}
OFormer	5.020×10^{-1}	1.080×10^{0}	5.620×10^{-1}	7.620×10^{-1}
AROMA	$\mathbf{1.480 \times 10^{-2}}$	$\mathbf{2.780 \times 10^{-2}}$	$\mathbf{5.720 \times 10^{-2}}$	$\mathbf{1.940 \times 10^{-1}}$

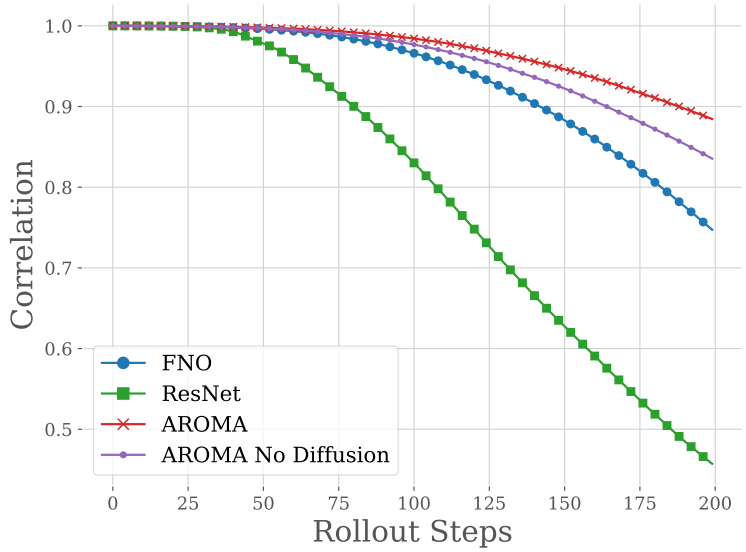


Figure 7.3: Correlation over time for long rollouts with different methods on *Burgers*.

only trained to predict the next state. We plot the correlation over rollout steps of different methods, including our model without the diffusion process, in Figure 7.3. We can clearly see the gain in stability in using the diffusion for long rollouts. Still, the predictions will eventually become uncorrelated over time as the solver accumulates errors compared with the numerical solution. As we employ a generative model, we can generate several rollouts and estimate the uncertainty of the solver with standard deviations. We can see in Appendix Figure B.8 that this uncertainty increases over time. This uncertainty is not a guarantee that the solution lies within the bounds, but is an indication that the model is not confident in its predictions.

7.5 Related Work

Our model differs from existing models in the field of operator learning and more broadly from existing neural field architectures. The works most related to ours are the following.

Neural Fields for PDE Neural Fields have recently emerged as powerful tools to model dynamical systems. DINO (Yin et al., 2022b) is a space-time continuous architecture based on a modulated multiplicative filter network (Fathony et al., 2021b) and a NeuralODE (Chen and Zhang, 2019) for modeling the dynamics. DINO is capable of encoding and decoding physical states on irregular grids thanks to the spatial continuity of the INR and through auto-decoding (Park et al., 2019a). CORAL is another neural-field based architecture, which tackles the broader scope of operator learning, also builds on meta-learning (Zintgraf et al., 2019b; Dupont et al., 2022a) to freely process irregular grids. CORAL and DINO are the most similar works to ours, as they are both auto-regressive and capable of processing irregular grids. On the other hand Chen et al. (2022) and Hagnberger et al. (2024) make use of spatio-temporal Neural Fields, for obtaining smooth and compact latent representations in the first or to directly predict trajectory solutions within a temporal horizon in the latter. Moreover, they either use a CNN or rely on patches for encoding the observations and are therefore not equipped for the type of tasks AROMA is designed for.

Transformers for PDE Several PDE solvers leverage transformers and cross-attention as a backbone for modeling PDEs. Transformers, which operate on token sequences, provide a natural solution for handling irregular meshes and point sets. Li et al. (2023a) and Hao et al. (2023) introduced transformer architectures tailored for operator learning. Hao et al. (2023) incorporated an attention mechanism and employed a mixture of experts strategy to address multi-scale challenges. However, their architecture relies on linear attention without reducing spatial dimensions, resulting in linear complexity in sequence size, but quadratic in the hidden dimensions, which can be prohibitive for deep networks and large networks. Similarly, Li et al. (2023a) utilized cross-attention to embed both regular and irregular meshes into a latent space and applied a recurrent network for time-marching in this latent space. Nonetheless, like GNOT, their method operates point-wise on the latent space. Transolver (Wu et al., 2024) decomposes a discrete input function into a mixture of "slices," each corresponding to a prototype in a mixture model, with attention operating in this latent space. This approach, akin to our model, reduces complexity. However, it has not been designed for temporal problems. (Alkin et al., 2024) recently proposed a versatile model capable of operating on Eulerian and Lagrangian (particles) representations. They reduce input dimensionality by ag-

gregating information from input values onto "supernodes" selected from the input mesh via message passing while decoding is performed with a Perceiver-like architecture. In contrast, AROMA performs implicit spatial encoding with cross-attention to encode the geometry and aggregate obsevation values. Finally, their training involves complex end-to-end optimization, whereas we favor two simple training steps that are easier to implement.

7.6 Conclusion and Limitations

AROMA offers a novel and flexible neural operator approach for modeling the spatio-temporal evolution of physical processes. It is able to deal with general geometries and to forecast at any position of the spatial domain. It incorporates in an encode-process-decode framework attention mechanisms, a latent diffusion transformer for spatio-temporal dynamics and neural fields for decoding. Thanks to a very compact spatial encoding, its complexity is lower than most SOTA models. Experiments with small-size datasets demonstrate its effectiveness. Its reduced complexity holds potential for effective scaling to larger datasets. As for the limitations, the performance of AROMA are still to be demonstrated on larger and real world examples. Moreover, like all dynamical models that operate over a latent space, the reconstruction capabilities of the decoder is a bottleneck for the rollout accuracy. Since the encoder and decoder are learning spatial relationships from scratch, conferring the framework a high flexibility, the training efficiency does not match that of CNN-based auto-encoders on regular grids. We therefore believe there could be further improvements to be made to achieve a similar performance while keeping the same level of flexibility. Finally, even though our model has some potential for uncertainty modeling, this aspect has still to be further explored and analyzed.

Chapter 8

Zebra: In-Context and Generative Pretraining for Solving Parametric PDEs

Solving time-dependent parametric partial differential equations (PDEs) is challenging for data-driven methods, as these models must adapt to variations in parameters such as coefficients, forcing terms, and initial conditions. State-of-the-art neural surrogates perform adaptation through gradient-based optimization and meta-learning to implicitly encode the variety of dynamics from observations. This often comes with increased inference complexity. Inspired by the in-context learning capabilities of large language models (LLMs), we introduce `Zebra`, a novel generative autoregressive transformer designed to solve parametric PDEs without requiring gradient adaptation at inference. By leveraging in-context information during both pre-training and inference, `Zebra` dynamically adapts to new tasks by conditioning on input sequences that incorporate context example trajectories. As a generative model, `Zebra` can be used to generate new trajectories and allows quantifying the uncertainty of the predictions. We evaluate Zebra across a variety of challenging PDE scenarios, demonstrating its adaptability, robustness, and superior performance compared to existing approaches.

Serrano, L., Koupaï, A. K., Wang, T. X., Erbacher, P., Gallinari, P. (2024). Zebra: In-Context and Generative Pretraining for Solving Parametric PDEs. arXiv preprint.

8.1 Introduction

A major challenge for training neural solvers for time dependent partial differential equations (PDEs) or more generally for modeling spatio-temporal dynamics is to capture the variety of behaviors arising from complex physical phenomena. In particular, neural solvers, trained from a limited number of situations often fail to generalize to new physical contexts and situations (Chen et al., 2018; Raissi et al., 2019; Li et al., 2021; Koupaï et al., 2024).

We address the parametric PDE problem (Cohen and Devore, 2015), where the goal is to train models on a limited set of scenarios representing a given physical phenomenon so that they can generalize across a wide range of new contexts, including different PDE parameters. These parameters may encompass initial and boundary conditions, physical coefficients, and forcing terms. In this work, we focus on purely data-driven approaches that do not incorporate prior knowledge of the underlying equations.

A basic approach to this problem is to sample from the distribution of physical parameters, i.e., to train on different instances of a PDE characterized by varying parameter values, with the goal of generalizing to unseen instances. This approach relies on an i.i.d. assumption and requires a training set that adequately represents the distribution of the underlying dynamical system—a condition that is often difficult to satisfy in practice due to the complexity of physical phenomena. Other approaches explicitly condition on specific PDE parameters (Brandstetter et al., 2022d; Takamoto et al., 2023), relying on the availability of such prior knowledge. This assumes that a physical model of the observed system is known, making the incorporation of PDE parameters into neural solvers challenging beyond physical

quantities. Moreover, in many cases, this prior knowledge is incomplete or entirely unknown.

An alternative approach involves adaptation to new PDE instances by leveraging observations from novel *environments*. Here we consider that an environment is characterized by a set of parameters. For data-driven models, adaptation is often performed through fine tuning, which usually requires a significant amount of examples for the new environment. This is for example the setting adopted in many recent development of foundation models (Subramanian et al., 2023; Herde et al., 2024; McCabe et al., 2023). This involves training a large model on a variety of physics-based numerical simulations, each requiring a large amount of simulations with the expectation that it will generalize.

More principled frameworks for adaptation leverage meta-learning, where the model is trained on simulations corresponding to different environments—i.e., varying PDE parameter values—so that it can quickly adapt to new and unseen PDE simulation instances using a few trajectory examples (Park et al., 2023; Kirchmeyer et al., 2022; Yin et al., 2022a). These flexible methods rely on gradient updates for adaptation, adding computational overhead.

We explore a new direction for adaptation inspired by the successes of in-context learning (ICL) in natural language processing (NLP) and its demonstrated ability to generalize to downstream tasks without retraining (Brown et al., 2020; Touvron et al., 2023). We propose a framework, denoted `Zebra`, relying on ICL for solving parametric PDEs with new parameter values, without any additional update of the model parameters.

As for ICL in NLP, the model is trained to generate appropriate responses given context examples and a query. The context examples will be trajectories from the same dynamics starting from different initial conditions. The query will consist for example of a new initial state condition, that will serve as inference starting point for the new forecast. The proposed model is inspired from NLP approaches: it is a causal generative model that processes discrete token sequences encoding observations. It is trained to model the trajectory distributions of parameteric PDEs. This approach offers key advantages and greater flexibility compared to existing methods. It can leverage contexts of different types and sizes, requires only a few context examples to adapt to new dynamics, and allows us to cover a wide range of situations. It provides enhanced capabilities compared to more classical deterministic forecasting models. Notably, generative probabilistic models have been developed for physical problems such as weather forecasting Price et al. (2025); Couairon et al. (2024a) and even PDE solving Lippe et al. (2023), demonstrating superior performance and capabilities over their deterministic counterparts. However, their setting is different, as they rely on diffusion models and are neither designed for adaptation nor intended to address the parametric PDE problem.

Some recent works have also explored adaptation through in-context learning for

dynamics modeling. The closest to ours is probably Yang et al. (2023), which also targets adaptation to multiple environments of an underlying physical dynamics through prompting with examples. Their model employs a specific deterministic encoder-decoder architecture and is limited to 1D ODEs or sparse 2D data due to scalability issues. More details and further references are provided in Appendix C.1.

On the technical side, `Zebra` introduces a novel generative autoregressive solver for parametric PDEs. It employs an encode-generate-decode framework: first, a vector-quantized variational auto-encoder (VQ-VAE) (Oord et al., 2017) is learned to compress physical states into discrete tokens and to decode it back to the original physical space. Next, a generative autoregressive transformer is pre-trained with arbitrary size context examples of trajectories using a next token objective. At inference, `Zebra` can handle varying context sizes for conditioning and supports uncertainty quantification, enabling generalization to unseen PDE parameters without gradient updates.

Our main contributions include:

- We introduce a generative autoregressive transformer for modeling physical dynamics. It operates on compact discretized representations of physical state observations. This framework represents the first successful application of causal generative modeling using quantized representations of physical systems.

- To harness the in-context learning strengths of autoregressive transformers, we develop a new pretraining strategy that conditions the model on example trajectories with similar dynamics but different initial conditions.

- Our generative model predicts trajectory distributions. This provides a richer information than deterministic auto-regressive models. This comes with enhanced capabilities including more accurate predictions, uncertainty measures, or the ability to sample and generate new trajectories conditioned on some examples.

- `Zebra` scales better than gradient based adaptation approaches. Furthermore, we propose an accelerated inference procedure that scales order of magnitude faster than other adaptation methods.

- We evaluate `Zebra` against adaptation methods in a one-shot setting. The model infers dynamics from a context trajectory that shares similar behavior with the target but differs in initial conditions, representing a one-shot setting. `Zebra`'s performance is benchmarked against domain-adaptation baselines specifically trained for such tasks.

8.2 Problem setting

8.2.1 Solving parametric PDEs

We aim to solve parametric time-dependent PDEs beyond the typical variation in initial conditions. Our goal is to train models capable of generalizing across a wide range of PDE parameters. To this end, we consider time-dependent PDEs with different initial conditions, and with additional degrees of freedom, namely: (1) coefficient parameters — such as fluid viscosity or advection speed — denoted by vector μ ; (2) boundary conditions $\mathcal{B}$, e.g. Neumann or Dirichlet; (3) forcing terms δ, including damping parameter or sinusoidal forcing with different frequencies. We denote $\xi := \{\mu, \mathcal{B}, \delta\}$ and we define $\mathcal{F}_\xi$ as the set of PDE solutions corresponding to the PDE parameters μ, boundary conditions $\mathcal{B}$ and forcing term δ, and refer to $\mathcal{F}_\xi$ as a PDE environment. Formally, a solution $\boldsymbol{u}(x,t)$ within $\mathcal{F}_\xi$ satisfies:

$$\begin{aligned} \frac{\partial \boldsymbol{u}}{\partial t} &= F\big(\delta, \mu, t, x, \boldsymbol{u}, \frac{\partial \boldsymbol{u}}{\partial x}, \frac{\partial^2 \boldsymbol{u}}{\partial x^2}, \dots\big), \\ &\qquad \forall x \in \Omega, \forall t \in (0,T] \\ \mathcal{B}(\boldsymbol{u})(x,t) &= 0, \quad \forall x \in \partial\Omega, \forall t \in (0,T] \\ \boldsymbol{u}(0,x) &= \boldsymbol{u}^0, \quad \forall x \in \Omega \end{aligned} \tag{8.1}$$

where F is a function of the solution $\boldsymbol{u}$ and its spatial derivatives on the domain Ω, and also includes the forcing term δ ; $\mathcal{B}$ is the boundary condition constraint (e.g., spatial periodicity, Dirichlet, or Neumann) that must be satisfied at the boundary of the domain $\partial\Omega$; and $\boldsymbol{u}^0$ is the initial condition sampled with a probability measure $\boldsymbol{u}^0 \sim \boldsymbol{p}^0(.)$.

8.2.2 Adaptation for parametric PDE

Solving time-dependent parametric PDEs requires developing neural solvers capable of generalizing to a whole distribution of PDE parameters. In practice, changes in the PDE parameters often lead to distribution shifts in the trajectories which makes the problem challenging. Different directions are currently being explored and are briefly reviewed below. We focus on pure data-driven approaches that do not make use of any prior knowledge on the equations. We make the assumption that the models are learned from numerical simulations so that it is possible to generate from multiple parameters. This emulates real situations where for example, a physical phenomenon is observed in different contexts.

Fine tuning pre-trained models The classical strategy for adapting to new settings is to fine tune models that have been pretrained on a distribution of the PDE parameters. This approach often relies on large fine tuning samples and involves

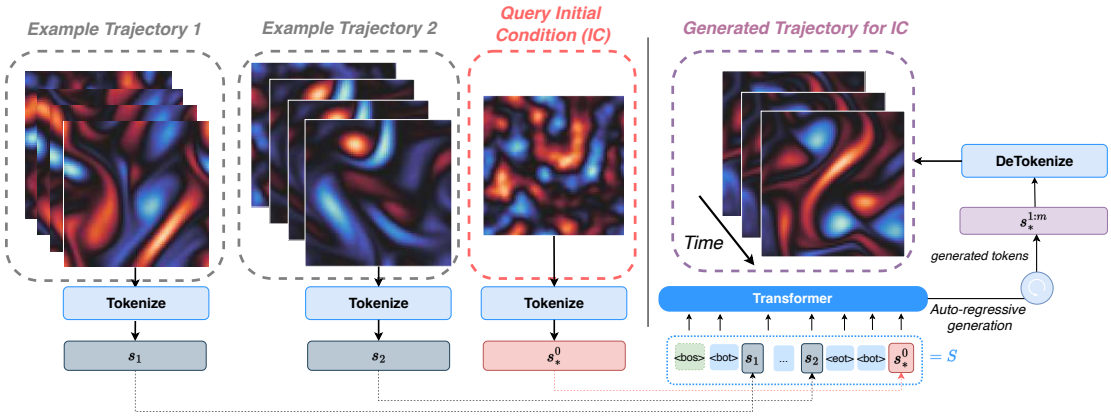


Figure 8.1: `Zebra`'s inference pipeline from context trajectories. The model leverages a set of example trajectories (u_1 and u_2), which share similar dynamics but have different initial conditions, to generate a sequence that follows the same underlying physical behavior but coming from a new initial condition (query) u_*^0. The example trajectories and the initial condition are first tokenized into index sequences s_1, s_2 and s_*^0, which are then concatenated according to the sequence design used during pretraining. The transformer autoregressively predicts the next tokens to complete the sequence. Finally, the generated indices are detokenized to reconstruct the solution in the physical space.

updating all or a subset of parameters (Subramanian et al., 2023; Herde et al., 2024). We do not consider this option that has been shown to underperform SOTA adaptation approaches Koupaï et al. (2024).

Gradient-based adaptation A more flexible approach relies on adaptation at inference time through meta-learning. It posits that a set of environments e are available from which trajectories are sampled, each environment e being defined by specific PDE parameter values (Zintgraf et al., 2019a; Kirchmeyer et al., 2022). The model is trained from a sampling from the environments distribution to adapt fast to a new environment. The usual formulation is to learn shared and specific environment parameters $\mathcal{G}_{\theta+\theta_\xi}$, where θ and θ_ξ are respectively the shared and specific parameters. At inference, for a new environment, only a small number of parameters θ_ξ is adapted from a small sample of observations. This family of method will be our reference baseline in the following.

8.3 Zebra Framework

We introduce `Zebra`, a novel framework designed to solve parametric PDEs through in-context learning and flexible conditioning. `Zebra` utilizes a generative autoregressive transformer to model partial differential equations (PDEs) within a compact,

discrete latent space. A spatial CNN encoder is employed to map physical spatial observations into these latent representations, while a CNN decoder accurately reconstructs them. Our pretraining pipeline consists of two key stages: 1) Learning a finite vocabulary of physical phenomena, and 2) Training the transformer using an in-context pretraining strategy, enabling the model to effectively condition on contextual information. At inference, `Zebra` allows to perform in-context learning from context trajectories as illustrated in Figure 8.1.

8.3.1 Learning a finite vocabulary of physical phenomena

In order to leverage the auto-regressive transformer architecture and adopt a next-token generative pretraining, we need to convert physical observations into discrete representations. We do not quantize the observations directly but rather quantize compressed latent representations by employing a VQVAE (Oord et al., 2017), an encoder-decoder architecture with a quantizer component. Our encoder spatially compresses the input function $\boldsymbol{u}^t$ through a convolutional model $\mathcal{E}_w$, which maps the input to a continuous latent variable $\mathbf{z}^t = \mathcal{E}_w(\boldsymbol{u}^t)$. The latent variables are then quantized to a vector of discrete codes $\mathbf{z}_q^t$ using a codebook $\mathcal{Z}$ of size $K = |\mathcal{Z}|$ through the quantization component q. For each spatial code $\mathbf{z}_{[ij]}^t$ in $\mathbf{z}_q^t$, the nearest codebook entry z_k is selected. The decoder $\mathcal{D}_\psi$ reconstructs the signal $\hat{\boldsymbol{u}}^t$ from the quantized latent codes $\hat{\mathbf{z}}_q^t$. Both models are jointly trained to minimize the reconstruction error between the function $\boldsymbol{u}^t$ and its reconstruction $\hat{\boldsymbol{u}}^t = \mathcal{D}_\psi \circ q \circ \mathcal{E}_w(\boldsymbol{u}^t)$.

Once this training step is done, we can tokenize a trajectory $\boldsymbol{u}^{t:t+m\Delta t}$ by applying our encoder in parallel on each timestamp to obtain vectors of discrete codes $\mathbf{z}_q^{t:t+m\Delta t}$ and retrieve the corresponding index entries $s^{t:t+m\Delta t}$ from the codebook $\mathcal{Z}$. Similarly, we detokenize discrete indices with the decoder. We provide a brief description of the VQVAE model and details on its architecture in Appendix C.3.

8.3.2 In-context modeling

We design sequences that enable `Zebra` to perform in-context learning on trajectories that share underlying dynamics with different initial states. To incorporate varying amounts of contextual information, we draw a number $n \in \{1, n_{\max}\}$, then sample n trajectories sharing the same dynamics, each with m snapshots starting from time t, denoted as $(\boldsymbol{u}_1^{t:t+m\Delta t}, \ldots, \boldsymbol{u}_n^{t:t+m\Delta t})$. These trajectories are tokenized into index representations $(s_1^{t:t+m\Delta t}, \ldots, s_n^{t:t+m\Delta t})$, which are flattened into sequences $s_1, \ldots, s_n$, maintaining the temporal order from left to right. In practice, we fix $n_{\max} = 6$ and $m = 9$.

Since our model operates on tokens from a codebook, we found it advantageous to introduce *special tokens* to structure the sequences. The tokens `<bot>` (beginning of trajectory) and `<eot>` (end of trajectory) define the boundaries of each trajec-

tory within the sequence. Furthermore, as we sample sequences with varying context sizes, we maximize the utilization of the transformer's context window by stacking sequences that could also represent different dynamics. To signal that these sequences should not influence each other, we use the special tokens `<bos>` (beginning of sequence) and `<eos>` (end of sequence). The final sequence design is:

$$S = \texttt{<bot>}[s_1]\texttt{<eot>} \ldots \texttt{<bot>}[s_n]\texttt{<eot>}$$

And our pretraining dataset is structured as follows:

$$\texttt{<bos>}[S_1]\texttt{<eos>} \ldots \texttt{<bos>}[S_l]\texttt{<eos>}$$

8.3.3 Next-token pretraining

The transformer is trained using self-supervised learning on a next-token prediction task with teacher forcing (Radford et al., 2018). Given a sequence S of discrete tokens of length N, the model is optimized to minimize the negative log-likelihood (cross-entropy loss):

$$\mathcal{L}_{\text{Transformer}} = -\mathbb{E}_S \sum_{i=1}^{N} \log p(S_{[i]} | S_{[i'<i]}),$$

where the model learns to predict each token $S_{[i]}$ conditioned on all previous tokens $S_{[i'<i]}$. Due to the transformer's causal structure, earlier tokens in the sequence are not influenced by later ones, while later tokens benefit from more context, allowing for more accurate predictions. This structure naturally supports both generation in a one-shot and few-shot setting within a unified framework. Our transformer implementation is based on the Llama architecture (Touvron et al. (2023)). Additional details can be found in Appendix C.3. Up to our knowledge, this is the first adaptation of generative auto-regressive transformers to the modeling of physical dynamics.

8.3.4 Flexible inference: prompting and sampling

In this section, we outline the inference pipeline for `Zebra` across various scenarios. For simplicity, we assume that all observations have already been tokenized and omit the detokenization process. Let s_* represent the target token sequence to be predicted.

- **Prompting** with n examples and an *initial condition*: The prompt is structured as $S = \texttt{<bos><bot>}[s_1^{0:m\Delta t}] \ldots [s_n^{0:m\Delta t}]\texttt{<eot><bot>}[s_*^0]$, allowing the model to adapt based on the provided examples and initial condition.

- **Prompting** with n examples and ℓ *frames*: This setup combines context from multiple trajectories with the initial timestamps, structured as $S =$ `<bos><bot>`$[s_1^{0:m\Delta t}] \dots [s_n^{0:m\Delta t}]$`<bot>`$[s_*^{0:\ell\Delta t}]$.

At inference, we adjust the *temperature parameter* τ of the classifier layer to calibrate the level of diversity of the next-token distributions. The temperature τ scales the logits y_i before the softmax function :

$$p(S_{[i]} = k | S_{[i'<i]}) = \text{softmax}\left(\frac{y_k}{\tau}\right) = \frac{\exp\left(\frac{y_k}{\tau}\right)}{\sum_j \exp\left(\frac{y_j}{\tau}\right)}$$

When $\tau > 1$, the distribution becomes more uniform, encouraging exploration, whereas $\tau < 1$ sharpens the distribution, favoring more deterministic predictions. During training, it is kept fixed at $\tau = 1$.

8.4 Experiments

In this section, we experimentally validate that our framework enables one-shot adaptation at inference. We follow the *pretraining* procedure outlined in Section 8.3 for each dataset described in Section 8.4.1 and evaluate `Zebra` across distinct scenarios. We first assess its performance in the one-shot setting for in-distribution parameters, comparing it to adaptation-based baselines (Section 8.4.2). We then examine its generalization in out-of-distribution settings in Section 8.4.3. Next we illustrate and analyze the generative abilities of `Zebra` through two example tasks: *uncertainty quantification* and *new trajectory generation* in Section 8.4.4, with further analysis on this topic in Appendix C.4.2 and Appendix C.4.3. Finally, in Section 8.4.5, we show how we can drastically accelerate the adaptation at inference compared to gradient-based methods. Additional results can be found Appendix C.4.

8.4.1 Datasets details

As in Kirchmeyer et al. (2022), we generate data in batches, where each batch of trajectories corresponds to a single environment and shares the same PDE parameters while having different initial conditions. We consider various factors of variation across multiple datasets. To assess the generalization ability of our model across a wide range of scenarios, we use a significantly larger number of environments—far exceeding those in previous studies and available simulation datasets (Yin et al. (2022a), Kirchmeyer et al. (2022), Blanke and Lelarge (2023), Nzoyem et al. (2024)). We conduct experiments across seven datasets: five 1D—*Advection*, *Heat*, *Burgers*, *Wave-b*, *Combined*—and two 2D—*Wave 2D*, *Vorticity*. These datasets were selected to encompass different physical phenomena and test generalization under changes to various types of PDE parameters, as described below.

Table 8.1: One-shot adaptation. Conditioning from a similar trajectory. Test results in relative L2 on the trajectory. '–' indicates inference has diverged.

	Advection	*Heat*	*Burgers*	*Wave b*	*Combined*	*Wave 2D*	*Vorticity 2D*
CAPE	0.00941	0.223	0.213	0.978	**0.00857**	–	–
CODA	**0.00687**	0.546	0.767	1.020	0.0120	0.777	<u>0.678</u>
`[CLS]` ViT	0.140	**0.136**	<u>0.116</u>	<u>0.971</u>	0.0446	<u>0.271</u>	0.972
ViT-`in-context`	0.0902	0.472	0.582	0.472	0.0885	0.390	0.173
`Zebra`	<u>0.00794</u>	<u>0.154</u>	**0.115**	**0.245**	<u>0.00965</u>	**0.207**	**0.119**

Varying PDE coefficients The changing factor is the set of coefficients μ in Equation (1). For *Burgers*, *Heat*, and *Vorticity 2D* equations, the viscosity coefficient ν varies across environments. For *Advection*, the advection speed β changes. In *Wave-c* and *Wave-2D*, the wave's celerity c is unique to each environment, and the damping coefficient k varies across environments in *Wave-2D*. In the *Combined* equation, three coefficients (α, β, γ) vary, each influencing different derivative terms respectively: $-\frac{\partial u^2}{\partial x}, +\frac{\partial^2 u}{\partial x^2}, -\frac{\partial^3 u}{\partial x^3}$ on the right-hand side of Equation (8.1).

Varying boundary conditions In this case, the varying parameter is the boundary condition $\mathcal{B}$ from Equation (8.1). For *Wave-b*, we explore two types of boundary conditions—Dirichlet and Neumann—applied independently to each boundary, resulting in four distinct environments.

Varying forcing term The varying parameter is the forcing term δ in Equation (1). In *Burgers* and *Heat*, the forcing terms vary by the amplitude, frequency, and shift coefficients of $\delta(t, x) = \sum_{j=1}^{5} A_j \sin\left(\omega_j t + 2\pi \frac{l_j x}{L} + \phi_j\right)$.

A detailed description of the datasets is provided in Appendix C.2, while Table C.1 summarizes the number of environments used during training, the number of trajectories sharing the same dynamics, and the varying PDE parameters across environments. For testing, all methods are evaluated on trajectories with new initial conditions in previously unseen environments. These unseen environments include trajectories with both novel initial conditions and varying parameters, which remain within the training distribution for in-distribution evaluation and extend beyond it for out-of-distribution testing. For each testing, we use 120 unseen environments for the 2D datasets and 12 for the 1D datasets, with each environment containing 10 trajectories.

8.4.2 In-distribution generalization

Setting We evaluate `Zebra`'s ability to perform in-context learning by *leveraging example trajectories that follow the same underlying dynamics as the target.* For-

mally, in the n-shot adaptation setting, we assume access to a set of n context trajectories $\{\boldsymbol{u}_1^{0:m\Delta t}, \ldots, \boldsymbol{u}_n^{0:m\Delta t}\}$ at inference time, all of which belong to the same dynamical system $\mathcal{F}_\xi$. The goal of the adaptation task is to accurately predict a future trajectory $\boldsymbol{u}_*^{\Delta t:m\Delta t}$ from a new initial condition $\boldsymbol{u}_*^0$, knowing that the underlying target dynamics is shared with the provided context example trajectories.

Sampling For `Zebra`, we use here a random sampling procedure at inference for generating the next tokens for all datasets, setting a low temperature ($\tau = 0.1$) to prioritize accuracy over diversity. Predictions are generated using a single sample under this configuration.

Baselines We evaluate Zebra against 4 baselines, CODA (Kirchmeyer et al., 2022) and CAPE (Takamoto et al., 2023), two SOTA adaptation methods. We also compare to two specifically designed ViT architectures: `[CLS]` ViT that performs adaptation by learning a `[CLS]` embedding and ViT-`in-context` designed for in-context training. CODA is a meta-learning framework designed for learning parametric PDEs. It leverages common knowledge from multiple environments where trajectories from a same environment e share the same PDE parameter values. CODA training performs adaptation in the parameter space by learning shared parameters across all environments and a context vector c^e specific to each environment. At inference, CODA adapts to a new environment by tuning c^e with several gradient steps. CAPE was not designed to perform adaptation via extra-trajectories, but instead needs the correct parameter values as input to condition a neural solver. We adapt it to our setting, by learning a context c^e instead of using the real parameter values. During adaptation, we only tune this context c^e via gradient updates. `[CLS]` ViT is a specifically designed baseline based on a vision transformer (Peebles and Xie, 2023), integrating a `[CLS]` token that serves as a learned parameter for each environment. This token lets the model handle different dynamics, and during inference, we adapt the `[CLS]` vector via gradient updates, following the same approach used in CODA and CAPE. ViT-`in-context` is a transformer with separate temporal and spatial attention (Ho et al., 2019), where we stack context examples and preceding target frames in the temporal axis to provide in-context examples. Note that all these baselines are deterministic.

Metrics We evaluate the performance using the Relative L^2 norm between the predicted rollout trajectory $\hat{\boldsymbol{u}}_*^{\text{trajectory}}$ and the ground truth $\boldsymbol{u}_*^{\text{trajectory}}$: $L^2_{\text{test}} = \frac{1}{N_{\text{test}}} \sum_{j \in \text{test}} \frac{||\hat{u}_j^{\text{trajectory}} - u_j^{\text{trajectory}}||_2}{||u_j^{\text{trajectory}}||_2}$.

Results As evidenced in Table 8.1, `Zebra` demonstrates strong overall performance in the one-shot adaptation setting, often surpassing gradient-based adaptation methods. For the more challenging datasets, such as *Burgers*, *Wave-b*, and

the 2D cases, `Zebra` consistently achieves lower relative L2 errors, highlighting its capacity to model complex dynamics effectively. Notably, `Zebra` excels in 2D environments, outperforming both CODA and `[CLS]`ViT and avoiding the divergence issues encountered by CAPE. While `Zebra` performs comparably to CODA on simpler datasets like *Advection* and *Combined*, its overall stability and versatility across a range of scenarios, particularly in 2D settings, highlight its competitiveness. Overall, `Zebra` stands out as a reliable and scalable solution for adaptation for solving parametric PDEs, demonstrating that in-context learning offers a robust alternative to existing gradient-based adaptation methods. The experiments also demonstrate a first benefit of the generative ability of `Zebra`. Deterministic models tend to predict conditional "blurry" expectations so that errors accumulate during the autoregressive process (see Appendix C.4.1 for more details). Zebra on his side has been trained to learn trajectory distributions, and is able to sample from this distribution at inference, generating more precise trajectories, less prone to error accumulation.

8.4.3 Out-of-distribution generalization

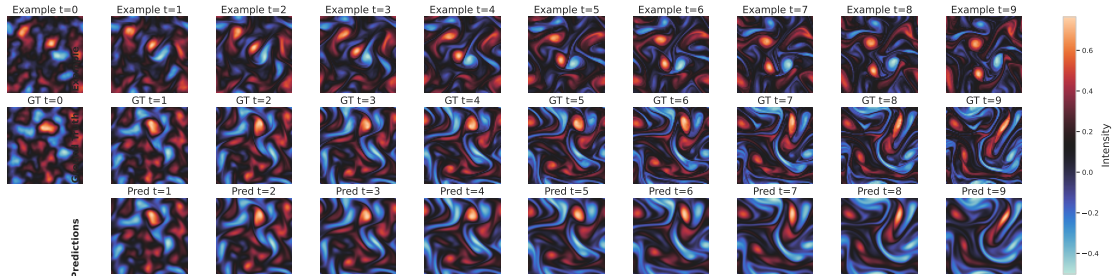


Figure 8.2: One-shot prediction on *Vorticity* in the turbulent OoD regime $\nu \in [1e-5, 1e-4]$. Top-row is the example given in context, mid-row is the ground truth trajectory, and the bottom-row is the generation with `Zebra`.

Datasets We evaluate here our models on new PDE instances, focusing on the following distribution shifts: (i) *Heat*: The forcing coefficients are varied from Appendix C.2.3, with parameters sampled as $A_j \in [-1.0, 1.0]$, $\omega_j \in [-0.8, 0.8]$ compared to the training ranges $[-0.5, 0.5]$ and $[-0.4, 0.4]$; (ii) *Vorticity 2D*: In the *close* setting, we sample viscosity within $[5 \times 10^{-4}, 10^{-3}]$, whereas during training, it was sampled from $[10^{-3}, 10^{-2}]$. We also evaluate a *far* setting, shifting to a more turbulent regime with viscosity sampled in $[10^{-5}, 10^{-4}]$, see Figure 8.2; (iii) *Wave 2D*: The wave celerity c is sampled from $[500, 550]$ (compared to $[100, 500]$ for training), and the damping term k is sampled from $[50, 60]$ (compared to $[0, 50]$ for training).

Setting We evaluate all the models in a one-shot setting on trajectories with out-of-distribution PDE parameters on new initial conditions, making this a particularly

challenging test of generalization.

Results We report the scores in Table 8.2. Overall, all methods experience performance degradation due to the distribution shift, with `Zebra` achieving the best results in three out of four experiments, while CODA and CAPE perform the worst. This poor performance for CAPE and CODA is expected on the 2D datasets, as they already struggled to generalize within the training distribution. However, for the *Heat* equation, errors for CAPE and CODA double, whereas `Zebra` maintains similar accuracy, demonstrating greater robustness to distribution shifts. Comparing `Zebra` and ViT-`in-context` to CAPE and CODA, it is remarkable that adaptation through in-context learning appears to be a more effective alternative than gradient-based adaptation for out-of-distribution generalization.

Out-of-distribution generalization remains a challenging task, particularly under strong shifts. On the *Vorticity* dataset, `Zebra` adapts to large shifts in viscosity and predicts the large-scale component of the dynamics. As shown in Figure 8.2, the predictions are not as sharp as the ground truths, as the VQVAE was not explicitly trained to capture the part of the spectrum present in turbulent trajectories.

Table 8.2: Out-of-distribution results. Test results in relative L2 on the trajectory. '–' indicates inference has diverged.

	Heat	*Wave 2D*	*Vorticity 2D*	
			close	*far*
CAPE	0.47	–	–	–
CODA	1.03	1.51	1.71	–
ViT-`in-context`	0.52	**0.68**	0.30	0.368
`Zebra`	**0.15**	**0.68**	**0.24**	**0.317**

8.4.4 Generative ability of the model

The evaluation in Section 8.4.2 already shows that as a generative model, `Zebra` is less prone to error accumulation that deterministic auto-regressive models. We illustrate here additional benefits from the generative capabilities of `Zebra` through two example tasks: *uncertainty quantification* and *new trajectory generation*. Further analysis of the behavior of `Zebra` is provided in Appendix C.4.2 and Appendix C.4.3.

Uncertainty quantification Given a context example and an initial condition, `Zebra` can generate multiple trajectories thanks to the sampling operation at the classifier level. Statistics can then be derived from this sample of the trajectory distribution in order to assess for example the uncertainty associated to a prediction. An illustration is provided in Figure 8.3, the red curve represents the ground truth,

the blue curve is the predicted mean and the blue shading indicates the empirical confidence interval (3 × standard deviation). Mean and standard deviation are calculated pointwise. We provide a more detailed analysis in Appendix C.4.2. In particular, it shows as expected that (i) uncertainty can be calibrated via the temperature parameter τ (Figure C.13), and (ii) it decreases with additional context (Table C.4).

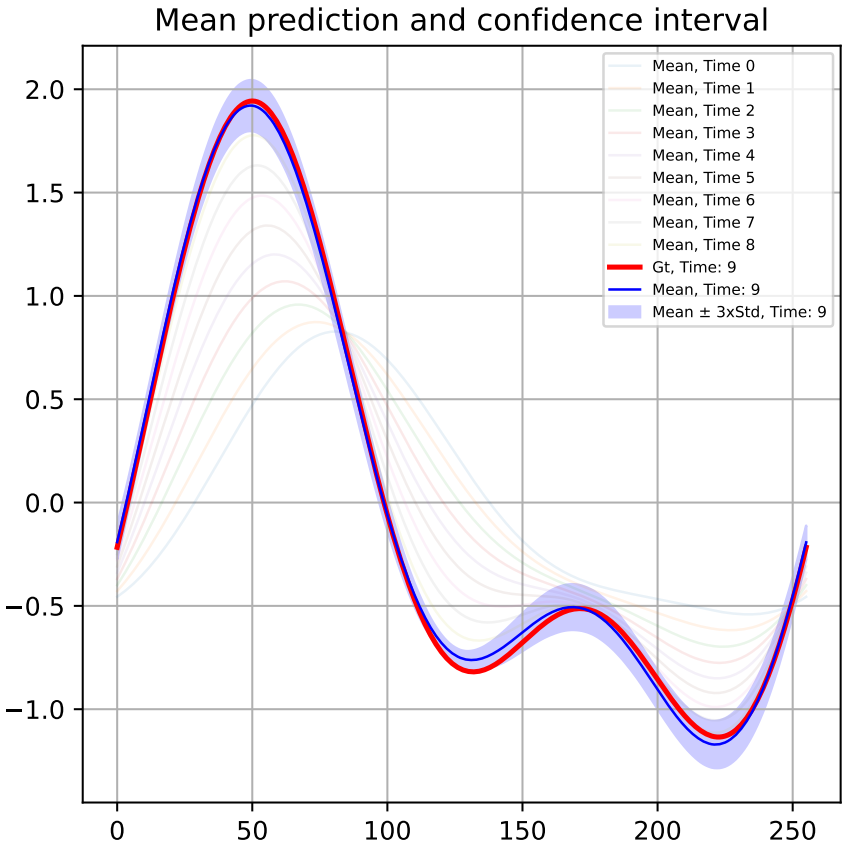
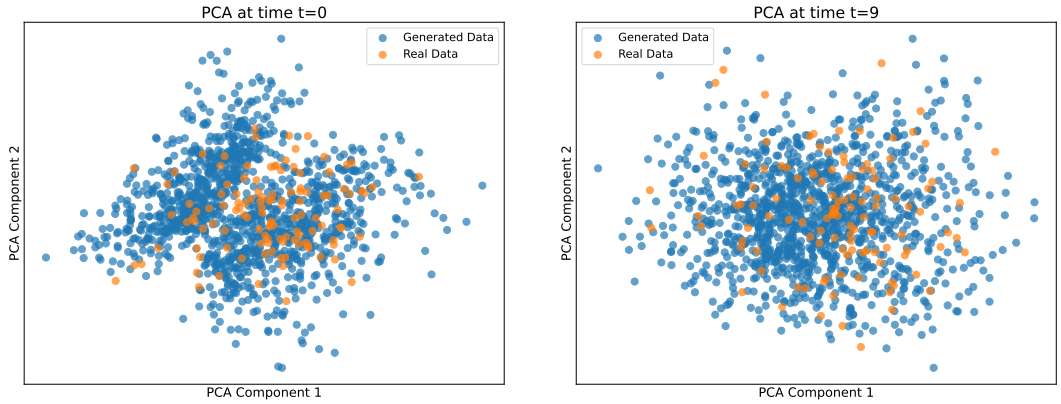


Figure 8.3: Uncertainty quantification with `Zebra` in a one-shot setting on *Heat* equation.

New trajectory generation As a second illustration, we assess `Zebra`'s ability to generate relevant new trajectories conditioned on in-context examples alone, i.e. without prompting with an initial state query. This is similar to conditional image or text generation in LLMs. The generated trajectories are sampled conditioned on a context trajectory from a new unseen environment. The key finding here is that `Zebra` effectively generates faithful trajectory distributions that closely match the real simulated trajectory distribution. Hence, given some examples from an unknown environment, `Zebra` could be used to generate trajectories that comply with the distribution in this environment. Qualitatively, Figure 8.4 illustrates how the real (from a held out sample) and the generated distributions match at two different

time steps ($t = 0, t = 9$): the PCA projection, indicates a strong alignment. Quantitatively, Table 8.3 shows that the Wasserstein distance of the generated trajectories is comparable to the Wasserstein distance between validation and test samples. As a calibration measure, we also provide in Table 8.3 the Wasserstein distance between the real distribution and a Gaussian distribution. Further details are provided in Appendix C.4.3.



$t = 0$. $t = 9$.

Figure 8.4: PCA Visualization of generated (blue) vs. real (orange) trajectories on *Combined Equation*.

Table 8.3: Comparison of distributions using the Wasserstein distance between generated trajectories and real trajectories.

Distance Metric	*Advection*	*Combined Equation*
Gaussian noise vs. real data	18.22	16.15
Validation data vs. test data	5.11	1.87
`Zebra`-generated data vs. real data	5.57	2.21

8.4.5 Accelerating inference

Since `Zebra` requires no gradient steps at inference, it is already faster than gradient-based adaptation (see Table 8.4). However, its autoregressive nature introduces significant overhead at inference: generating trajectories token by token increases solver calls by one to two orders of magnitude compared to direct surrogate modeling, making inference costly. To address this, we propose a fast inference method that accelerates inference by orders of magnitude relative to both the original model and gradient-based adaptation (Table 8.4). Instead of token-wise autoregressive generation, we predict entire frames at once. This is achieved by replacing the token-wise

autoregressive generation process with a frame-wise autoregressive surrogate, implemented as a UNet. The UNet, conditioned on a context embedding output by the transformer, takes frame $\boldsymbol{u}^t$ as input and predicts $\hat{\boldsymbol{u}}^{t+\Delta t}$. A crucial component is the context embedding, which captures underlying dynamics from example trajectories. This is learned by introducing a new token, `[DYN]`, at the transformer's input, analogous to `[CLS]` in BERT, allowing attention to encode context dynamics effectively. The implementation is detailed in Appendix C.3 and in Figure C.3. Table 8.4 shows that this reduces inference time by one to two orders of magnitude, making `Zebra` highly efficient.

Table 8.4: Inference times for one-shot adaptation. Average time in seconds to predict a single trajectory given a context trajectory and an initial condition. Times include adaptation and forecast for CODA and CAPE, while for `Zebra`, they include encoding, autoregressive prediction, and decoding.

	Advection	*Vorticity 2D*
CAPE	18s	23s
CODA	31s	28s
Zebra	3s	21s
Zebra + UNet	**0.10s**	**0.14s**

As shown in Table 8.5, this framework matches or outperforms pretrained `Zebra` in most cases. The dynamics embedding captures meaningful context, enabling efficient UNet training. In contrast, methods like CAPE and CODA must learn both model weights and environment embeddings simultaneously, making training less efficient.

Table 8.5: `Zebra` vs `Zebra` + UNet, for the in-distribution one-shot setting. Test results in relative L2 error.

	Advection	*Combined*	*Wave 2D*	*Vorticity 2D*
`Zebra`	0.00794	**0.00965**	0.207	0.119
`Zebra`+UNet	**0.0072**	0.0138	**0.150**	**0.0869**

8.5 Limitations

The quality of the generated trajectories is limited by the decoder's ability to reconstruct details from the quantized latent space. While reconstructions are excellent for many applications, scaling the codebook size (Yu et al., 2023a; Mentzer et al., 2023) or exploring alternatives to vector quantization (Li et al., 2024b) could further enhance reconstruction, provided in-context capability is preserved. Additionally,

replacing convolutional blocks with more flexible architectures, such as those in Serrano et al. (2024b), could extend the model to irregularly sampled or complex systems.

8.6 Conclusion

This study introduces `Zebra`, a novel generative model that adapts language model pretraining techniques for solving parametric PDEs. We propose a pretraining strategy that enables `Zebra` to develop in-context learning capabilities. Our experiments demonstrate that the pretrained model performs competitively against specialized baselines across various scenarios. Additionally, as a generative model, `Zebra` facilitates uncertainty quantification and can generate new trajectories, providing valuable flexibility in applications.

Part III

Conclusion

Chapter 9

Conclusion

We have now reached the final chapter of the core section of this thesis. In what follows, I summarize the main contributions presented throughout this manuscript and reflect on their limitations and the research directions they open for the future.

9.1 Synthesis

In this thesis, we have developed several novel frameworks for training surrogate models from numerical simulations. Our work addresses key limitations of prior deep learning approaches to solving partial differential equations, through the design of four methods: CORAL, Infinity, AROMA, and Zebra. In particular, we introduced new frameworks for learning meshless operators in the context of PDE modeling, and proposed a novel approach to enable in-context adaptation, fostering generalization to previously unseen dynamics.

9.1.1 Meshless operators

- *Handling irregular domains*: We proposed two distinct approaches for handling irregular domains: CORAL and AROMA. Both can process obervations on irregular grids and allow a continuous spatial representation of the PDE solutions but differ in how they encode observations, process and decode features. CORAL relies on autodecoding, which is elegant but computationally intensive during both training and inference, it also struggles with complex dynamics. In contrast, AROMA employs attention blocks at all stages, making it more effective for challenging dynamics. Both methods generalize across different discretization levels and adapt to various geometries.

- *Encoding domain geometry*: We explored two different strategies to encode domain geometry. In Infinity, we explicitly encoded the signed distance function (SDF) into a latent representation, which was then used by a process

network for geometry-aware inference. In AROMA, we adopted a different approach: a set of tokens attends to the positions of all observations to encode domain geometry. While the first approach requires preprocessing to extract SDF features, the second can be applied directly to any new geometry.

- *Temporal extrapolation*: In CORAL and AROMA, the dynamics evolve in a structured latent space with key properties. CORAL learns a compact and smooth latent representation that facilitates temporal extrapolation. AROMA, while less compact, maintains spatial structure, enabling more stable long-term predictions. Additionally, in AROMA we demonstrated that using a denoising objective for training improves stability and allows for longer temporal extrapolation.

9.1.2 In-context adaptation for solving PDEs

- *Adaptive solutions at inference*: We introduced Zebra, a framework that enables neural surrogates to dynamically adapt to variations in PDE coefficients, forcing terms, and boundary conditions, without requiring finetuning or gradient-based adaptation. Zebra leverages in-context learning through autoregressive transformers, a paradigm originally developed for language modeling. By reformulating PDE modeling as a sequence modeling task, Zebra achieves strong generalization across a wide range of physical systems. In particular, it outperforms existing meta-learning approaches for PDEs in the setting of one-shot adaptation.

- *Uncertainty quantification*: Both AROMA and Zebra are generative models, allowing for the sampling of probable trajectories conditioned on observations. In AROMA, we showed that the standard deviation of predictions could serve as a measure of uncertainty, providing information on the reliability of the model. In Zebra, we further quantified the uncertainty through relative L2 loss, standard deviation analysis, and confidence intervals. This uncertainty can be potentially calibrated via the temperature parameter and is sensitive to the amount of contextual information provided.

9.2 Limitations

Our contributions, while meaningful for the field, also have limitations.

Neural Fields Autoencoder While AROMA and CORAL are designed to handle irregular grids, they do not perform as efficiently on regular grids as classical CNN autoencoders. This presents a challenge, as an ideal model should generalize

across diverse dynamics and geometries within a unified framework. This limitation also suggests that there is room for improvement in the encoding and decoding mechanisms of neural fields and transformers when applied to irregular grids.

Discrete Tokenization In Zebra, the tokenization process is discrete, enabling the use of efficient autoregressive transformers originally designed for language modeling. However, this approach has certain limitations. Continuous tokenizers are generally known to yield better reconstruction performance than their discrete counterparts. In the context of surrogate modeling, we conducted preliminary experiments using a continuous VAE-based tokenizer, coupled with a transformer that models distributions as a mixture of Gaussians on relatively simple datasets Serrano et al. (2024a). This approach remains preliminary and requires further investigation.

9.3 Perspectives

Deep learning for physical simulation is still in its early stages, and given the complexity of the tasks, many challenges remain. Building on the limitations discussed above, I believe that addressing the following aspects could have a significant impact on the field.

A More Efficient Autoencoder The approaches introduced with AROMA and Zebra, both relying on different autoencoder architectures, open promising avenues for PDE modeling. Developing an autoencoder that matches the efficiency of CNNs on regular grids while simultaneously improving performance on irregular grids would be a major breakthrough in the field. Such an architecture would enable a universal model capable of encoding and decoding data from diverse numerical simulations and potentially across different geometries. Furthermore, improving autoencoder architectures could also impact frameworks based on the encode-process-decode paradigm.

Transformers with Continuous Tokens Developing adaptive methods like Zebra that leverage continuous tokenization could enhance accuracy and mitigate the limitations of discrete tokenization. Unlike images, physical signals often lack predefined ranges, making discretization a potential bottleneck in surrogate modeling. Addressing this challenge would require replacing Zebra's discrete autoregressive transformer with an alternative architecture that operates on continuous representations. A promising research direction is to explore the interplay between diffusion models and autoregression, investigating how these generative paradigms can be effectively combined to model continuous physical phenomena. This could lead to more expressive and flexible representations, reducing the information loss introduced by token quantization.

Self-Supervised Learning for Physics Early work on foundation models for physics (McCabe et al. (2023)) has primarily relied on temporal autoregressive pre-training. These models are first trained on a given dataset and can be fine-tuned on others, demonstrating improved generalization over training from scratch. Although effective, this approach requires fine-tuning large models, which can be computationally expensive and limit scalability.

An alternative strategy would be to train a smaller model that leverages meaningful features extracted from a foundation model, acting as a backbone for downstream tasks. In computer vision, this principle is exemplified by foundation models such as DINO (Caron et al., 2021) and CLIP (Radford et al., 2021), which, unlike large language models, do not generate data but rather extract high-level semantic information from images. Applying a similar approach to physics could be particularly useful for modeling physical dynamics, allowing models to capture underlying physical parameters from trajectories or to infer the influence of boundary conditions.

Determining whether such foundation models can be effectively obtained for physics remains an open question. Although some preliminary works, such as Mialon et al. (2023), have explored self-supervised learning in this domain, the direction is still largely unexplored. Unlike vision models, which often rely on data augmentations, this approach may be suboptimal for physics-based modeling, as arbitrary transformations may not preserve the underlying physical structure. A promising direction is to draw inspiration from the recent work of Darcet et al. (2025), which eliminates the need for data augmentation altogether. This raises the exciting possibility that similar self-supervised methodologies could be extended to other modalities, potentially unlocking new avenues for self-supervised learning in physics.

Bibliography

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *ArXiv*, abs/1603.04467, 2016. URL https://api.semanticscholar.org/CorpusID:5707386.

J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A learning algorithm for boltzmann machines. *Cognitive science*, 9(1):147–169, 1985.

N. Agarwal, A. Ali, M. Bala, Y. Balaji, E. Barker, T. Cai, P. Chattopadhyay, Y. Chen, Y. Cui, Y. Ding, et al. Cosmos world foundation model platform for physical ai, 2025.

B. Alkin, A. Fürst, S. Schmid, L. Gruber, M. Holzleitner, and J. Brandstetter. Universal Physics Transformers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL http://arxiv.org/abs/2402.12365.

K. R. Allen, T. Lopez-Guevara, K. Stachenfeld, A. Sanchez-Gonzalez, P. Battaglia, J. Hamrick, and T. Pfaff. Physical design using differentiable learned simulators, 2 2022. URL http://arxiv.org/abs/2202.00728.

A. Arakawa. Computational design for long-term numerical integration of the equations of fluid motion: Two-dimensional incompressible flow. part i. *Journal of Computational Physics*, 1(1):119–143, 1966. ISSN 0021-9991. doi: https://doi.org/10.1016/0021-9991(66)90015-5. URL https://www.sciencedirect.com/science/article/pii/0021999166900155.

J. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *ArXiv*, abs/1607.06450, 2016. URL https://api.semanticscholar.org/CorpusID:8236317.

A. Baevski, Y. Zhou, A. Mohamed, and M. Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in neural information processing systems*, 33:12449–12460, 2020.

D. Bahdanau. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

G. A. Baker. Essentials of padé approximants, 1975. URL https://api.semanticscholar.org/CorpusID:118128264.

F. Bartolucci, E. de Bézenac, B. Raonic, R. Molinaro, S. Mishra, and R. Alaifari. Representation equivalent neural operators: a framework for alias-free operator learning. *Advances in Neural Information Processing Systems*, 36, 2024.

M. Bauer, E. Dupont, A. Brock, D. Rosenbaum, J. Schwarz, and H. Kim. Spatial functa: Scaling functa to imagenet classification and generation. *CoRR*, abs/2302.03130, 2023.

S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. *CoRR*, abs/1506.03099, 2015. URL http://arxiv.org/abs/1506.03099.

E. D. Bézenac, A. Pajot, and P. Gallinari. Deep learning for physical processes: incorporating prior scientific knowledge. *Journal of Statistical Mechanics: Theory and Experiment*, 2019, 2017. URL https://api.semanticscholar.org/CorpusID:2808403.

C. M. Bishop and N. M. Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

M. Blanke and M. Lelarge. Interpretable meta-learning of physical systems. *arXiv preprint arXiv:2312.00477*, 2023.

F. Bonnet, J. A. Mazari, P. Cinnella, and P. Gallinari. Airfrans : High fidelity computational fluid dynamics dataset for approximating reynolds-averaged navier – stokes solutions. In *Neurips 2022*, 2022.

L. L. Boudec, E. De Bézenac, L. Serrano, R. D. Regueiro-Espino, Y. Yin, and P. Gallinari. Learning a neural solver for parametric pde to enhance physics-informed methods. In *International Conference on Learning Representations*, 2025.

K. D. Brabanter, J. D. Brabanter, B. D. Moor, and I. Gijbels. Derivative estimation with local polynomial fitting. *J. Mach. Learn. Res.*, 14:281–301, 2013. URL https://api.semanticscholar.org/CorpusID:19018224.

J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/jax-ml/jax.

J. Brandstetter, R. van den Berg, M. Welling, and J. K. Gupta. Clifford neural layers for pde modeling. In *The Eleventh International Conference on Learning Representations*, 2022a.

J. Brandstetter, M. Welling, and D. E. Worrall. Lie point symmetry data augmentation for neural pde solvers. *ArXiv*, abs/2202.07643, 2022b. URL https://api.semanticscholar.org/CorpusID:246863584.

J. Brandstetter, M. Welling, and D. E. Worrall. Lie point symmetry data augmentation for neural pde solvers. In *International Conference on Machine Learning*, pages 2241–2256. PMLR, 2022c.

J. Brandstetter, D. E. Worrall, and M. Welling. Message passing neural pde solvers. *International Conference on Learning Representations*, 2022d.

M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners, 2020.

S. L. Brunton and J. N. Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019.

K. J. Burns, G. M. Vasil, J. S. Oishi, D. Lecoanet, and B. P. Brown. Dedalus: A flexible framework for numerical simulations with spectral methods. *Physical Review Research*, 2, 2020. ISSN 26431564. doi: 10.1103/PhysRevResearch.2.023068.

S. Cai, Z. Mao, Z. Wang, M. Yin, and G. E. Karniadakis. Physics-informed neural networks (pinns) for fluid mechanics: a review. *Acta Mechanica Sinica/Lixue Xuebao*, 37, 2021. ISSN 16143116. doi: 10.1007/s10409-021-01148-1.

Z. Cai, H. Zhu, Q. Shen, X. Wang, and X. Cao. Batch normalization alleviates the spectral bias in coordinate networks. *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 25160–25171, 2024. URL https://api.semanticscholar.org/CorpusID:272724642.

M. Cameron. Notes on burgers' equation, 2016. URL https://www.math.umd.edu/~mariakc/burgers.pdf. Accessed: [17/02/2025].

J. R. Cannon. *The One-Dimensional Heat Equation*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1984.

Y. Cao, Y. Liu, L. Yang, R. Yu, H. Schaeffer, and S. Osher. Vicon: Vision in-context operator networks for multi-physics fluid dynamics prediction, 11 2024. URL http://arxiv.org/abs/2411.16063.

M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9650–9660, 2021.

R. Caruana. Multitask learning. *Machine Learning*, 28:41–75, 1997. URL https://api.semanticscholar.org/CorpusID:45998148.

G. Catalani, S. Agarwal, X. Bertrand, F. Tost, M. Bauerheim, and J. Morlier. Neural fields for rapid aircraft aerodynamics simulations. *Scientific Reports*, 14, 2024. URL https://api.semanticscholar.org/CorpusID:271533723.

S. Chae, J. Shin, S. Kwon, S. Lee, S. Kang, and D. Lee. Pm10 and pm2.5 real-time prediction models using an interpolated convolutional neural network. *Scientific Reports*, 11, 2021. ISSN 20452322. doi: 10.1038/s41598-021-91253-9.

N. Chalapathi, Y. Du, and A. Krishnapriyan. Scaling physics-informed hard constraints with mixture-of-experts. *ArXiv*, abs/2402.13412, 2024. URL https://api.semanticscholar.org/CorpusID:267770212.

H. Chang, H. Zhang, L. Jiang, C. Liu, and W. T. Freeman. Maskgit: Masked generative image transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11315–11325, 2022.

M. Chen, A. Radford, R. Child, J. Wu, H. Jun, D. Luan, and I. Sutskever. Generative pretraining from pixels. In *International conference on machine learning*, pages 1691–1703. PMLR, 2020a.

P. Y. Chen, J. Xiang, D. H. Cho, Y. Chang, G. A. Pershing, H. T. Maia, M. Chiaramonte, K. Carlberg, and E. Grinspun. Crom: Continuous reduced-order modeling of pdes using implicit neural representations. *International Conference on Learning Representation*, 6 2022. URL http://arxiv.org/abs/2206.02607.

R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

R. T. Q. Chen. torchdiffeq, 2018. URL https://github.com/rtqichen/torchdiffeq.

T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020b.

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.

Y. Chen and X. Wang. Transformers as meta-learners for implicit neural representations. In *European Conference on Computer Vision*, 2022. URL https://api.semanticscholar.org/CorpusID:251320414.

Z. Chen and H. Zhang. Learning implicit fields for generative shape modeling. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June, 2019. ISSN 10636919. doi: 10.1109/CVPR.2019.00609.

Clay Mathematics Institute. The millennium prize problems, 2000. URL https://www.claymath.org/millennium-problems. Accessed: [17/02/2025].

D.-A. Clevert. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

A. Cohen and R. Devore. Approximation of high-dimensional parametric pdes. *Acta Numerica*, 2015.

F. Cole, Y. Lu, R. O'Neill, and T. Zhang. Provable in-context learning of linear systems and linear elliptic pdes with transformers. *arXiv preprint arXiv:2409.12293*, 2024.

C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20:273–297, 1995.

G. Couairon, R. Singh, A. Charantonis, C. Lessig, and C. Monteleoni. Archesweather and archesweathergen: a deterministic and generative model for efficient ml weather forecasting, 12 2024a. URL http://arxiv.org/abs/2412.12971.

P. Couairon, M. Shukor, J.-E. Haugeard, M. Cord, and N. Thome. Diffcut: Catalyzing zero-shot semantic segmentation with diffusion features and recursive normalized cut. In *Neural Information Processing Systems*, 2024b. URL https://api.semanticscholar.org/CorpusID:270258028.

R. Courant, K. Friedrichs, and H. Lewy. On the partial difference equations of mathematical physics. *IBM Journal of Research and Development*, 11(2):215–234, 1967. doi: 10.1147/rd.112.0215.

K. Cranmer, J. Brehmer, and G. Louppe. The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences*, 117:30055 – 30062, 2019. URL https://api.semanticscholar.org/CorpusID:207871083.

R. M. Cummings, W. H. Mason, S. A. Morton, and D. R. McDaniel. *Applied computational aerodynamics: A modern engineering approach*, volume 53. Cambridge University Press, 2015.

M. Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. *Advances in neural information processing systems*, 26, 2013.

D. Dai, Y. Sun, L. Dong, Y. Hao, S. Ma, Z. Sui, and F. Wei. Why can gpt learn in-context? language models implicitly perform gradient descent as meta-optimizers. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 4005–4019, 2023. ISBN 9781959429623. doi: 10.18653/v1/2023.findings-acl.247.

T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

T. Darcet, F. Baldassarre, M. Oquab, J. Mairal, and P. Bojanowski. Cluster and predict latents patches for improved masked image modeling. *arXiv preprint arXiv:2502.08769*, 2025.

Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier. Language modeling with gated convolutional networks. In *International conference on machine learning*, pages 933–941. PMLR, 2017.

F. de Avila Belbute-Peres, T. D. Economon, and J. Z. Kolter. Combining differentiable pde solvers and graph neural networks for fluid flow prediction. In *International Conference on Machine Learning*, 2020. URL https://api.semanticscholar.org/CorpusID:220424832.

T. De Ryck, F. Bonnet, S. Mishra, and E. de Bézenac. An operator preconditioning perspective on training in physics-informed machine learning. *arXiv preprint arXiv:2310.05801*, 2023.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em - algorithm plus discussions on the paper. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977. URL https://api.semanticscholar.org/CorpusID:4193919.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Q. Dong, L. Li, D. Dai, C. Zheng, J. Ma, R. Li, H. Xia, J. Xu, Z. Wu, B. Chang, X. Sun, L. Li, and Z. Sui. A survey on in-context learning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1107–1128, Miami, Florida, USA, Nov. 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.64. URL https://aclanthology.org/2024.emnlp-main.64/.

A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

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. *International Conference on Learning Representations.*, 10 2021. URL http://arxiv.org/abs/2010.11929.

A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

E. Dupont, H. Kim, S. M. A. Eslami, D. Rezende, and D. Rosenbaum. From data to functa: Your data point is a function and you can treat it like one. *Proceedings of the 39 th International Conference on Machine Learning*, 1 2022a. URL http://arxiv.org/abs/2201.12204.

E. Dupont, H. Loya, M. Alizadeh, A. Goliński, Y. W. Teh, and A. Doucet. Coin++: Neural compression across modalities. *Transactions on Machine Learning Research*, 1 2022b. URL http://arxiv.org/abs/2201.12904.

P. Esser, R. Rombach, and B. Ommer. Taming transformers for high-resolution image synthesis, 2020.

P. Esser, R. Rombach, and B. Ommer. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12873–12883, 2021.

P. Esser, S. Kulal, A. Blattmann, R. Entezari, J. Muller, H. Saini, Y. Levi, D. Lorenz, A. Sauer, F. Boesel, D. Podell, T. Dockhorn, Z. English, K. Lacey, A. Goodwin, Y. Marek, and R. Rombach. Scaling rectified flow transformers for high-resolution image synthesis. *ArXiv*, abs/2403.03206, 2024. URL https://api.semanticscholar.org/CorpusID:268247980.

R. Eymard, T. Gallouët, and R. Herbin. Finite volume methods. *Handbook of numerical analysis*, 7:713–1018, 2000.

V. S. Fanaskov and I. V. Oseledets. Spectral neural operators. In *Doklady Mathematics*, volume 108, pages S226–S232. Springer, 2023.

R. Fathony, A. K. Sahu, D. Willmott, and J. Z. Kolter. Multiplicative filter networks. In *International Conference on Learning Representations*, 2021a.

R. Fathony, A. K. Sahu, D. Willmott, and J. Z. Kolter. Multiplicative filter networks. *International Conference on Learning Representations.*, 2021b.

J. Feldman. Solution of the heat equation by separation of variables, 2007. URL https://personal.math.ubc.ca/~feldman/m267/heatSln.pdf. Accessed: [17/02/2025].

J. H. Ferziger and M. Peric. *Computational methods for fluid dynamics*. Springer, 1996. URL https://api.semanticscholar.org/CorpusID:63211967.

M. Fey and J. E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

S. Flennerhag, A. A. Rusu, R. Pascanu, F. Visin, H. Yin, and R. Hadsell. Meta-learning with warped gradient descent. *arXiv preprint arXiv:1909.00025*, 2019.

J.-Y. Franceschi, A. Dieuleveut, and M. Jaggi. Unsupervised scalable representation learning for multivariate time series. *ArXiv*, abs/1901.10738, 2019. URL https://api.semanticscholar.org/CorpusID:59413908.

J. Galewsky, R. K. Scott, and L. M. Polvani. An initial-value problem for testing numerical models of the global shallow-water equations. *Tellus, Series A: Dynamic Meteorology and Oceanography*, 56, 2004. ISSN 02806495. doi: 10.1111/j.1600-0870.2004.00071.x.

H. Gao and S. Ji. Graph u-nets. In *international conference on machine learning*, pages 2083–2092. PMLR, 2019.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, 2010. URL https://api.semanticscholar.org/CorpusID:5575601.

T. Gneiting and A. E. Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102:359 – 378, 2007. URL https://api.semanticscholar.org/CorpusID:1878582.

G. H. Golub and C. H. Reinsch. Singular value decomposition and least squares solutions. *Numerische Mathematik*, 14:403–420, 1970. URL https://api.semanticscholar.org/CorpusID:123532178.

I. Goodfellow. Deep learning, 2016.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

C. Grossmann, H.-G. Roos, and M. Stynes. *Numerical treatment of partial differential equations*, volume 154. Springer, 2007.

G. Gupta, X. Xiao, and P. Bogdan. Multiwavelet-based operator learning for differential equations. *Advances in neural information processing systems*, 34:24048–24062, 2021.

J. K. Gupta and J. Brandstetter. Towards multi-spatiotemporal-scale generalized pde modeling. *Transactions on Machine Learning (TMLR) 07/2023*, 2022.

D. Ha, A. Dai, and Q. V. Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

J. Hagnberger, M. Kalimuthu, D. Musekamp, and M. Niepert. Vectorized Conditional Neural Fields: A Framework for Solving Time-dependent Parametric Partial Differential Equations. In *Proceedings of the 41st International Conference on Machine Learning (ICML 2024)*, 2024.

E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. Springer Series in Computational Mathematics, 2010. URL https://api.semanticscholar.org/CorpusID:117014104.

W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017a.

W. L. Hamilton. *Graph Representation Learning: Foundations, Methods, Applications and Systems*. Morgan and Claypool, 2020. ISBN 9781450383325. doi: 10.1145/3447548.3470824.

W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, volume 2017-December, 2017b.

Z. Hao, Z. Wang, H. Su, C. Ying, Y. Dong, S. Liu, Z. Cheng, J. Song, and J. Zhu. Gnot: A general neural operator transformer for operator learning. In *International Conference on Machine Learning*, pages 12556–12569. PMLR, 2023.

Z. Hao, C. Su, S. Liu, J. Berner, C. Ying, H. Su, A. Anandkumar, J. Song, and J. Zhu. Dpot: Auto-regressive denoising operator transformer for large-scale pde pre-training. *41th International Conference on Machine Learning (ICML 2024)*, 2024.

A. Hassani, S. Walton, J. Li, S. Li, and H. Shi. Neighborhood attention transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6185–6194, 2023.

T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015a. URL https://api.semanticscholar.org/CorpusID:206594692.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015b. URL https://api.semanticscholar.org/CorpusID:13740328.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-December, 2016a. ISSN 10636919. doi: 10.1109/CVPR.2016.90.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016b.

K. He, X. Chen, S. Xie, Y. Li, P. Doll'ar, and R. B. Girshick. Masked autoencoders are scalable vision learners. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15979–15988, 2021. URL https://api.semanticscholar.org/CorpusID:243985980.

D. Hendrycks and K. Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

M. Herde, B. Raonić, T. Rohner, R. Käppeli, R. Molinaro, E. de Bézenac, and S. Mishra. Poseidon: Efficient foundation models for pdes. In *Advances in Neural Information Processing Systems 38 (NeurIPS 2024)*, 2024.

G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

G. E. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *ArXiv*, abs/1503.02531, 2015. URL https://api.semanticscholar.org/CorpusID:7200347.

J. Ho, N. Kalchbrenner, D. Weissenborn, and T. Salimans. Axial attention in multidimensional transformers. *arXiv preprint arXiv:1912.12180*, 2019.

J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

J. Ho, T. Salimans, A. Gritsenko, W. Chan, M. Norouzi, and D. J. Fleet. Video diffusion models. *Advances in Neural Information Processing Systems*, 35:8633–8646, 2022.

S. Hochreiter, A. S. Younger, and P. R. Conwell. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*, 2001. URL https://api.semanticscholar.org/CorpusID:267810281.

E. Hopf. The partial differential equation ut + uux = μxx. *Communications on Pure and Applied Mathematics*, 3:201–230, 1950. URL https://api.semanticscholar.org/CorpusID:121837938.

J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982. doi: 10.1073/pnas.79.8.2554. URL https://www.pnas.org/doi/abs/10.1073/pnas.79.8.2554.

H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24:498–520, 1933. URL https://api.semanticscholar.org/CorpusID:144828484.

E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

A. Ibrahim, de Bézenac Emmanueland Pajot Arthur, and G. Patrick. Modelling spatiotemporal dynamics from Earth observation data with neural differential equations. *Machine Learning*, 111(6):2349–2380, 2022. ISSN 15730565. URL https://doi.org/10.1007/s10994-022-06139-2.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ArXiv*, abs/1502.03167, 2015. URL https://api.semanticscholar.org/CorpusID:5808102.

A. Jacot, F. Gabriel, and C. Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

L. Jiaqi, J. Cui, J. Yang, and B. Yang. Stochastic neural simulator for generalizing dynamical systems across environments, 08 2024.

C. Johnson. Finite element methods for flow problems, 1992. URL https://api.semanticscholar.org/CorpusID:118090226.

J. M. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis. Highly accurate protein structure prediction with alphafold. *Nature*, 596:583 – 589, 2021. URL https://api.semanticscholar.org/CorpusID:235959867.

T. Karras, M. Aittala, S. Laine, E. Härkönen, J. Hellsten, J. Lehtinen, and T. Aila. Alias-free generative adversarial networks. *Advances in neural information processing systems*, 34:852–863, 2021.

A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR, 2020.

B. Kerbl, G. Kopanas, T. Leimkuehler, and G. Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics (TOG)*, 42:1 – 14, 2023. URL https://api.semanticscholar.org/CorpusID:259267917.

G. Kerschen, J.-C. Golinval, A. F. Vakakis, and L. A. Bergman. The method of proper orthogonal decomposition for dynamical characterization and order reduction of mechanical systems: An overview. *Nonlinear Dynamics*, 41:147–169, 2005. URL https://api.semanticscholar.org/CorpusID:17625377.

P. Kidger. *On Neural Differential Equations*. PhD thesis, University of Oxford, 2021.

D. P. Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

M. Kirchmeyer, Y. Yin, J. Donà, N. Baskiotis, A. Rakotomamonjy, and P. Gallinari. Generalizing to new physical systems via context-informed dynamics model. In *International Conference on Machine Learning*, pages 11283–11301. PMLR, 2022.

D. Kochkov, J. A. Smith, A. Alieva, Q. Wang, M. P. Brenner, and S. Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21):e2101784118, 2021.

D. Kochkov, J. Yuval, I. Langmore, P. C. Norgaard, J. A. Smith, G. Mooers, M. Klöwer, J. Lottes, S. Rasp, P. D. Düben, S. Hatfield, P. W. Battaglia, A. Sanchez-Gonzalez, M. Willson, M. P. Brenner, and S. Hoyer. Neural general circulation models for weather and climate. *Nature*, 632:1060 – 1066, 2023. URL https://api.semanticscholar.org/CorpusID:265150435.

D. Kondratyuk, L. Yu, X. Gu, J. Lezama, J. Huang, R. Hornung, H. Adam, H. Akbari, Y. Alon, V. Birodkar, Y. Cheng, M.-C. Chiu, J. Dillon, I. Essa, A. Gupta, M. Hahn, A. Hauth, D. Hendon, A. Martinez, D. C. Minnen, D. A. Ross, G. Schindler, M. Sirotenko, K. Sohn, K. Somandepalli, H. Wang, J. Yan, M. Yang, X. Yang, B. Seybold, and L. Jiang. Videopoet: A large language model for zero-shot video generation. *ArXiv*, abs/2312.14125, 2023. URL https://api.semanticscholar.org/CorpusID:266435847.

A. K. Koupaï, J. M. Benet, Y. Yin, J.-N. Vittaut, and P. Gallinari. Geps: Boosting generalization in parametric pde neural solvers through adaptive conditioning. *arXiv preprint arXiv:2410.23889*, 2024.

N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, and A. Anandkumar. Journal of machine learning research. *arxiv*, 8 2021. URL http://arxiv.org/abs/2108.08481.

A. Krishnapriyan, A. Gholami, S. Zhe, R. Kirby, and M. W. Mahoney. Characterizing possible failure modes in physics-informed neural networks. *Advances in neural information processing systems*, 34:26548–26560, 2021.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60, 2017. ISSN 15577317. doi: 10.1145/3065386.

Y. Kuramoto. Diffusion-induced chaos in reaction systems. *Progress of Theoretical Physics Supplement*, 64:346–367, 1978. URL https://api.semanticscholar.org/CorpusID:122049161.

W. Kutta. Beitrag zur naherungsweisen integration totaler differentialgleichungen, 1901. URL https://api.semanticscholar.org/CorpusID:237071170.

E. Le Naour, G. Agoua, N. Baskiotis, and V. Guigue. Interpretable time series neural representation for classification purposes. In *2023 IEEE 10th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10, 2023. doi: 10.1109/DSAA60987.2023.10302534.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

D. Lee, C. Kim, M. Cho, and W.-S. Han. Locality-aware generalizable implicit neural representation. *Advances in Neural Information Processing Systems*, 2023.

K. Leung. Neural network architecture diagrams. https://github.com/kennethleungty/Neural-Network-Architecture-Diagrams, 2020. Accessed: 2025-03-26.

T. Li, Y. Tian, H. Li, M. Deng, and K. He. Autoregressive image generation without vector quantization. *ArXiv*, abs/2406.11838, 2024a. URL https://api.semanticscholar.org/CorpusID:270560593.

T. Li, Y. Tian, H. Li, M. Deng, and K. He. Autoregressive image generation without vector quantization. *arXiv preprint arXiv:2406.11838*, 2024b.

Z. Li, F. Zhou, F. Chen, and H. Li. Meta-sgd: Learning to learn quickly for few shot learning. *ArXiv*, abs/1707.09835, 2017a. URL https://api.semanticscholar.org/CorpusID:25316837.

Z. Li, F. Zhou, F. Chen, H. Li, T. Liu, and T.-S. Chen. Meta-sgd: Learning to learn quickly for few-shot learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2111–2120. JMLR. org, 2017b.

Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020a.

Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020b.

Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations. *International Conference on Learning Representations.*, 10 2021. URL http://arxiv.org/abs/2010.08895.

Z. Li, D. Z. Huang, B. Liu, and A. Anandkumar. Fourier Neural Operator with Learned Deformations for PDEs on General Geometries. In *arXiv:2207.05209v1*, 2022a.

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. *39th International Conference on Machine Learning (ICML2022)*, 11 2022b. URL http://arxiv.org/abs/2111.03794.

Z. Li, K. Meidani, and A. B. Farimani. Transformer for partial differential equations' operator learning. *Transactions on Machine Learning Research (April/2023)*, 2023a.

Z.-Y. Li, N. B. Kovachki, C. Choy, B. Li, J. Kossaifi, S. P. Otta, M. A. Nabian, M. Stadler, C. Hundt, K. Azizzadenesheli, and A. Anandkumar. Geometry-informed neural operator for large-scale 3d pdes. *ArXiv*, abs/2309.00583, 2023b. URL https://api.semanticscholar.org/CorpusID:261494027.

D. B. Lindell, D. V. Veen, J. J. Park, and G. Wetzstein. Bacon: Band-limited coordinate networks for multiscale scene representation. *Conference on Computer Vision and Pattern Recognition*, 12 2022. URL http://arxiv.org/abs/2112.04645.

Y. Lipman, R. T. Q. Chen, H. Ben-Hamu, M. Nickel, and M. Le. Flow matching for generative modeling. *ArXiv*, abs/2210.02747, 2022. URL https://api.semanticscholar.org/CorpusID:252734897.

P. Lippe, B. S. Veeling, P. Perdikaris, R. E. Turner, and J. Brandstetter. Pderefiner: Achieving accurate long rollouts with neural pde solvers. *arXiv preprint arXiv:2308.05732*, 2023.

H. Liu, D. Tam, M. Muqeeth, J. Mohta, T. Huang, M. Bansal, and C. A. Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems*, 35:1950–1965, 2022a.

X. Liu, C. Gong, and Q. Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *ArXiv*, abs/2209.03003, 2022b. URL https://api.semanticscholar.org/CorpusID:252111177.

Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9992–10002, 2021. URL https://api.semanticscholar.org/CorpusID:232352874.

Z. Long, Y. Lu, X. Ma, and B. Dong. Pde-net: Learning pdes from data. In *35th International Conference on Machine Learning, ICML 2018*, volume 7, 2018.

E. N. Lorenz. Deterministic nonperiodic flow. *Journal of the Atmospheric Sciences*, 20:130–141, 1963. URL https://api.semanticscholar.org/CorpusID:15359559.

A. J. Lotka. Elements of physical biology. *Nature*, 116:461–461, 1925. URL https://api.semanticscholar.org/CorpusID:4103581.

L. Lu, P. Jin, and G. E. Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *Nat Mach Intell*, 3:218–229, 10 2021a. doi: 10.1038/s42256-021-00302-5. URL http://arxiv.org/abs/1910.03193http://dx.doi.org/10.1038/s42256-021-00302-5.

L. Lu, R. Pestourie, W. Yao, Z. Wang, F. Verdugo, and S. G. Johnson. Physics-informed neural networks with hard constraints for inverse design. *SIAM Journal on Scientific Computing*, 43(6):B1105–B1132, 2021b.

L. Lu, X. Meng, S. Cai, Z. Mao, S. Goswami, Z. Zhang, and G. E. Karniadakis. *A comprehensive and fair comparison of two neural operators (with practical extensions) based on FAIR data*, volume 393. Elsevier B.V., 4 2022. doi: 10.1016/j.cma.2022.114778.

Y. Lu, M. Bartolo, A. Moore, S. Riedel, and P. Stenetorp. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. *ArXiv*, abs/2104.08786, 2021c. URL https://api.semanticscholar.org/CorpusID:233296494.

L. D. Luigi, A. Cardace, R. Spezialetti, P. Z. Ramirez, S. Salti, and L. di Stefano. Deep learning on implicit neural representations of shapes. *ArXiv*, abs/2302.05438, 2023. URL https://api.semanticscholar.org/CorpusID:256808600.

A. L. Maas. Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Ma- chine Learning*, 2013. URL https://api.semanticscholar.org/CorpusID:16489696.

J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, 1967. URL https://api.semanticscholar.org/CorpusID:6278891.

P. Marion, Y.-H. Wu, M. E. Sander, and G. Biau. Implicit regularization of deep residual networks towards neural odes. *ArXiv*, abs/2309.01213, 2023. URL https://api.semanticscholar.org/CorpusID:261530353.

M. McCabe, B. R.-S. Blancard, L. H. Parker, R. Ohana, M. Cranmer, A. Bietti, M. Eickenberg, S. Golkar, G. Krawezik, F. Lanusse, et al. Multiple physics pretraining for physical surrogate models. *arXiv preprint arXiv:2310.02994*, 2023.

F. Mentzer, D. Minnen, E. Agustsson, and M. Tschannen. Finite scalar quantization: Vq-vae made simple. *International Conference on Learning Representations*, 2023.

L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger. Occupancy networks: Learning 3d reconstruction in function space. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June, 2019. ISSN 10636919. doi: 10.1109/CVPR.2019.00459.

G. Mialon, Q. Garrido, H. Lawrence, D. Rehman, Y. LeCun, and B. T. Kiani. Self-supervised learning with lie symmetries for partial differential equations. *ArXiv*, abs/2307.05432, 2023. URL https://api.semanticscholar.org/CorpusID:259766697.

B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

S. Min, M. Lewis, L. Zettlemoyer, and H. Hajishirzi. Metaicl: Learning to learn in context. *arXiv preprint arXiv:2110.15943*, 2021.

S. Min, X. Lyu, A. Holtzman, M. Artetxe, M. Lewis, H. Hajishirzi, and L. Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? *ArXiv*, abs/2202.12837, 2022. URL https://api.semanticscholar.org/CorpusID:247155069.

R. M. Miura, C. S. Gardner, and M. D. Kruskal. Korteweg-de vries equation and generalizations. ii. existence of conservation laws and constants of motion. *Journal of Mathematical Physics*, 9:1204–1209, 1968. URL https://api.semanticscholar.org/CorpusID:121335529.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning, 12 2013.

A. T. Mohan, N. Lubbers, D. Livescu, and M. Chertkov. Embedding hard physical constraints in neural network coarse-graining of 3d turbulence. *arXiv preprint arXiv:2002.00021*, 2020.

K. W. Morton and D. F. Mayers. Numerical solution of partial differential equations: An introduction, 2005.

M. Mosbach, T. Pimentel, S. Ravfogel, D. Klakow, and Y. Elazar. Few-shot fine-tuning vs. in-context learning: A fair comparison and evaluation. *arXiv preprint arXiv:2305.16938*, 2023.

A. Muench. Partial differential equations (michaelmas term), 2023. URL https://courses.maths.ox.ac.uk/course/view.php?id=4986. Accessed: [17/02/2025].

T. Müller, A. Evans, C. Schied, and A. Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (TOG)*, 41: 1 – 15, 2022. URL https://api.semanticscholar.org/CorpusID:246016186.

T. Müller, A. Evans, C. Schied, and A. Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics*, 41(4):1–15, jul 2022. doi: 10.1145/3528223.3530127. URL https://doi.org/10.1145%2F3528223.3530127.

V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

G. Negiar, M. W. Mahoney, and A. S. Krishnapriyan. Learning differentiable solvers for systems with hard constraints. *ArXiv*, abs/2207.08675, 2022. URL https://api.semanticscholar.org/CorpusID:250627507.

A. Ng. Feature selection, l1 vs. l2 regularization, and rotational invariance. *Proceedings of the twenty-first international conference on Machine learning*, 2004. URL https://api.semanticscholar.org/CorpusID:11258400.

A. Nichol, J. Achiam, and J. Schulman. On first-order meta-learning algorithms, 2018.

R. D. Nzoyem, D. A. Barton, and T. Deakin. Neural context flows for meta-learning of dynamical systems. *arXiv preprint arXiv:2405.02154*, 2024.

O. Obiols-Sales, A. Vishnu, N. Malaya, and A. Chandramowliswharan. Cfdnet: A deep learning-based accelerator for fluid simulations. In *Proceedings of the 34th ACM international conference on supercomputing*, pages 1–12, 2020.

R. Ohana, M. McCabe, L. T. Meyer, R. Morel, F. J. Agocs, M. Beneitez, M. Berger, B. Burkhart, S. B. Dalziel, D. B. Fielding, D. Fortunato, J. A. Goldberg, K. Hirashima, Y.-F. Jiang, R. Kerswell, S. Maddu, J. M. Miller, P. Mukhopadhyay, S. S. Nixon, J. Shen, R. Watteaux, B. R.-S. Blancard, F. Rozet, L. H. Parker, M. Cranmer, and S. Ho. The well: a large-scale collection of diverse physics simulations for machine learning. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=00Sx577BT3.

T. OLMo, P. Walsh, L. Soldaini, D. Groeneveld, K. Lo, S. Arora, A. Bhagia, Y. Gu, S. Huang, M. Jordan, et al. 2 olmo 2 furious. *arXiv preprint arXiv:2501.00656*, 2024.

P. J. Olver. Introduction to partial differential equations, 2007. URL https://api.semanticscholar.org/CorpusID:52096086.

P. J. Olver. *Introduction to partial differential equations*. Undergraduate Texts in Mathematics. Springer Cham, 2014.

A. v. d. Oord, O. Vinyals, and K. Kavukcuoglu. Neural discrete representation learning. *arXiv preprint arXiv:1711.00937*, 2017.

OpenAI. Sora: Creating video from text, 2024. URL https://openai.com/sora. Accessed: 2024-01-16.

J. Pan, T. Gao, H. Chen, and D. Chen. What in-context learning "learns" in-context: Disentangling task recognition and task learning. In *Annual Meeting of the Association for Computational Linguistics*, 2023. URL https://api.semanticscholar.org/CorpusID:258740972.

J. Park, F. Berto, A. Jamgochian, M. Kochenderfer, and J. Park. First-order context-based adaptation for generalizing to new dynamical systems, 2023. URL https://openreview.net/forum?id=AW0i0l0hzqJ.

J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June, 2019a. ISSN 10636919. doi: 10.1109/CVPR.2019.00025.

J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 165–174, 2019b.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. *ArXiv*, abs/1912.01703, 2019a. URL https://api.semanticscholar.org/CorpusID:202786778.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019b. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

J. Pathak, S. Subramanian, P. Harrington, S. Raja, A. Chattopadhyay, M. Mardani, T. Kurth, D. Hall, Z. Li, K. Azizzadenesheli, et al. Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators. *arXiv preprint arXiv:2202.11214*, 2022.

G. Peano. Démonstration de l'intégrabilité des équations différentielles ordinaires. *Mathematische Annalen*, 37:182–228, 1890. URL https://api.semanticscholar.org/CorpusID:120698124.

W. Peebles and S. Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4195–4205, 2023.

E. Perez, F. Strub, H. D. Vries, V. Dumoulin, and A. Courville. Film: Visual reasoning with a general conditioning layer. *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, 2018a. ISSN 2159-5399. doi: 10.1609/aaai.v32i1.11671.

E. Perez, F. Strub, H. D. Vries, V. Dumoulin, and A. Courville. Film: Visual reasoning with a general conditioning layer. *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, 2018b. ISSN 2159-5399. doi: 10.1609/aaai.v32i1.11671.

T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. W. Battaglia. Learning mesh-based simulation with graph networks. *International Conference on Learning Representations.*, 10 2021. URL http://arxiv.org/abs/2010.03409.

B. Polyak and A. B. Juditsky. Acceleration of stochastic approximation by averaging. *Siam Journal on Control and Optimization*, 30:838–855, 1992. URL https://api.semanticscholar.org/CorpusID:3548228.

A. D. Polyanin. Handbook of linear partial differential equations for engineers and scientists, 2001. URL https://api.semanticscholar.org/CorpusID:203122873.

I. Price, A. Sanchez-Gonzalez, F. Alet, et al. Probabilistic weather forecasting with machine learning. *Nature*, 637:84–90, 2025. doi: 10.1038/s41586-024-08252-9. URL https://doi.org/10.1038/s41586-024-08252-9.

C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

A. M. Quarteroni and A. Valli. *Numerical Approximation of Partial Differential Equations*. Springer Series in Computational Mathematics, 2008. URL https://api.semanticscholar.org/CorpusID:120584359.

A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre-training, 2018.

A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners, 2019. URL https://www.semanticscholar.org/paper/Language-Models-are-Unsupervised-Multitask-Learners-Radford-Wu/9405cc0d6169988371b2755e573cc28650d14dfe.

A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378: 686–707, 2019.

P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

B. Raoni'c, R. Molinaro, T. D. Ryck, T. Rohner, F. Bartolucci, R. Alaifari, S. Mishra, and E. de B'ezenac. Convolutional neural operators for robust and accurate learning of pdes. In *Neural Information Processing Systems*, 2023. URL https://api.semanticscholar.org/CorpusID:258968120.

J. N. Reddy. *Introduction to the finite element method*. McGraw-Hill Education, 2019.

L. F. Richardson and P. Lynch. Weather prediction by numerical process, 1922. URL https://api.semanticscholar.org/CorpusID:62588316.

H. E. Robbins. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951. URL https://api.semanticscholar.org/CorpusID:16945044.

M. Rolinek, D. Zietlow, and G. Martius. Variational autoencoders pursue pca directions (by accident). In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12406–12415, 2019.

R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10674–10685, 2021. URL https://api.semanticscholar.org/CorpusID:245335280.

O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241. Springer, 2015a.

O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. *ArXiv*, abs/1505.04597, 2015b. URL https://api.semanticscholar.org/CorpusID:3719281.

F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

S. Rühling Cachay, B. Zhao, H. Joren, and R. Yu. DYffusion: a dynamics-informed diffusion model for spatiotemporal forecasting. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

C. Runge. Ueber die numerische auflösung von differentialgleichungen. *Mathematische Annalen*, 46:167–178, 1895. URL https://api.semanticscholar.org/CorpusID:119924854.

T. K. Rusch, M. M. Bronstein, and S. Mishra. A survey on oversmoothing in graph neural networks. *arXiv preprint arXiv:2303.10993*, 2023.

T. Salimans and J. Ho. Progressive distillation for fast sampling of diffusion models. *arXiv preprint arXiv:2202.00512*, 2022.

V. Saragadam, J. Tan, G. Balakrishnan, R. G. Baraniuk, and A. Veeraraghavan. Miner: Multiscale implicit neural representations, 2022.

L. Serrano, L. L. Boudec, A. K. Koupaï, T. X. Wang, Y. Yin, J.-N. Vittaut, and P. Gallinari. Operator learning with neural fields: Tackling pdes on general geometries. *Advances in Neural Information Processing Systems*, 2023.

L. Serrano, P. ERBACHER, J.-N. Vittaut, and patrick gallinari. Zebra: a continuous generative transformer for solving parametric PDEs. In *ICLR 2024 Workshop on AI4DifferentialEquations In Science*, 2024a. URL https://openreview.net/forum?id=KpBK1ArdXM.

L. Serrano, T. X. Wang, E. L. Naour, J.-N. Vittaut, and P. Gallinari. Aroma: Preserving spatial structure for latent pde modeling with local neural fields. *arXiv preprint arXiv:2406.02176*, 2024b.

N. Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

W. Shi, S. Min, M. Lomeli, C. Zhou, M. Li, G. Szilvasy, R. James, X. V. Lin, N. A. Smith, L. Zettlemoyer, et al. In-context pretraining: Language modeling beyond document boundaries. *arXiv preprint arXiv:2310.10638*, 2023.

J. Shue, E. Chan, R. Po, Z. Ankner, J. Wu, and G. Wetzstein. 3d neural field generation using triplane diffusion. *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20875–20886, 2022. URL https://api.semanticscholar.org/CorpusID:254095843.

D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. P. Lillicrap, K. Simonyan, and D. Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *ArXiv*, abs/1712.01815, 2017a. URL https://api.semanticscholar.org/CorpusID:33081038.

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. baker, M. Lai, A. Bolton, Y. Chen, T. P. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–359, 2017b. URL https://api.semanticscholar.org/CorpusID:205261034.

A. Simmons, D. M. Burridge, M. Jarraud, C. Girard, and W. Wergen. The ecmwf medium-range prediction models development of the numerical formulations and the impact of increased resolution. *Meteorology and Atmospheric Physics*, 40: 28–60, 1989. URL https://api.semanticscholar.org/CorpusID:121745059.

J. Sirignano and K. Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.

V. Sitzmann, E. R. Chan, R. Tucker, N. Snavely, and G. Wetzstein. Metasdf: Meta-learning signed distance functions. *Advances in Neural Information Processing Systems*, 2020-December, 2020a. ISSN 10495258.

V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33:7462–7473, 2020b.

V. Sitzmann, J. N. Martel, A. W. Bergman, D. B. Lindell, and G. Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 2020-December, 2020c. ISSN 10495258.

V. Sitzmann, J. N. P. Martel, A. W. Bergman, D. B. Lindell, and G. Wetzstein. Implicit neural representations with periodic activation functions. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020d.

J. Song, C. Meng, and S. Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

K. Stachenfeld, D. B. Fielding, D. Kochkov, M. Cranmer, T. Pfaff, J. Godwin, C. Cui, S. Ho, P. Battaglia, and A. Sanchez-Gonzalez. Learned coarse models for efficient turbulence simulation. *International Conference on Learning Representation)*, 2022.

J. C. Strikwerda. Finite difference schemes and partial differential equations, 1989. URL https://api.semanticscholar.org/CorpusID:120263575.

S. Subramanian, P. Harrington, K. Keutzer, W. Bhimji, D. Morozov, M. W. Mahoney, and A. Gholami. Towards foundation models for scientific machine learning: Characterizing scaling and transfer behavior. *Advances in Neural Information Processing Systems 37 (NeurIPS 2023)*, 2023.

R. Sutton. The bitter lesson. Blog post, 2019.

M. Takamoto, T. Praditia, R. Leiteritz, D. MacKinlay, F. Alesiani, D. Pflüger, and M. Niepert. Pdebench: An extensive benchmark for scientific machine learning. *Advances in Neural Information Processing Systems*, 35:1596–1611, 2022.

M. Takamoto, F. Alesiani, and M. Niepert. Learning neural pde solvers with parameter-guided channel attention. *International Conference on Machine Learning (ICML)*, 2023.

T. Takikawa, A. Evans, J. Tremblay, T. Müller, M. McGuire, A. Jacobson, and S. Fidler. Variable bitrate neural fields. *ACM SIGGRAPH 2022 Conference Proceedings*, 2022a. URL https://api.semanticscholar.org/CorpusID:249674957.

T. Takikawa, A. Evans, J. Tremblay, T. Müller, M. McGuire, A. Jacobson, and S. Fidler. Variable bitrate neural fields. *ACM Transactions on Graphics*, 2022b.

M. Tancik, P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. Barron, and R. Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, 33:7537–7547, 2020a.

M. Tancik, P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron, and R. Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, 2020-December, 2020b. ISSN 10495258.

M. Tancik, B. Mildenhall, T. Wang, D. Schmidt, P. P. Srinivasan, J. T. Barron, and R. Ng. Learned initializations for optimizing coordinate-based neural representations. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2021. ISSN 10636919. doi: 10.1109/CVPR46437.2021.00287.

N. Thuerey, K. Weißenow, L. Prantl, and X. Hu. Deep learning methods for reynolds-averaged navier–stokes simulations of airfoil flows. *AIAA Journal*, 58(1):25–36, 2020.

K. Tian, Y. Jiang, Z. Yuan, B. Peng, and L. Wang. Visual autoregressive modeling: Scalable image generation via next-scale prediction. In *Advances in neural information processing systems*, 2024.

Z. Tong, Y. Song, J. Wang, and L. Wang. Videomae: Masked autoencoders are data-efficient learners for self-supervised video pre-training. *Advances in neural information processing systems*, 35:10078–10093, 2022.

H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample. Llama: Open and efficient foundation language models. *CoRR*, 2023.

A. Tran, A. Mathews, L. Xie, and C. S. Ong. Factorized fourier neural operators. In *International Conference on Learning Representations*, 11 2023. URL http://arxiv.org/abs/2111.13802.

K. Um, R. Brand, Y. R. Fei, P. Holl, and N. Thuerey. Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers. *Advances in Neural Information Processing Systems*, 33:6111–6122, 2020a.

K. Um, Y. Fei, P. Holl, R. Brand, and N. Thürey. Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers. *ArXiv*, abs/2007.00016, 2020b. URL https://api.semanticscholar.org/CorpusID:220280657.

A. van den Oord, O. Vinyals, and K. Kavukcuoglu. Neural discrete representation learning. In *Advances in Neural Information Processing Systems 30*, pages 6306–6315, 2017.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Łukasz Kaiser, and I. Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017-December, 2017. ISSN 10495258.

P. Veličković, A. Casanova, P. Liò, G. Cucurull, A. Romero, and Y. Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 2018. doi: 10.1007/978-3-031-01587-8_7.

P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.

R. Vinuesa and S. L. Brunton. Enhancing computational fluid dynamics with machine learning. *Nature Computational Science*, 2(6):358–366, 2022.

V. Volterra. Variations and fluctuations of the number of individuals in animal species living together. *Ices Journal of Marine Science*, 3:3–51, 1928. URL https://api.semanticscholar.org/CorpusID:85082375.

J. von Oswald, E. Niklasson, E. Randazzo, J. Sacramento, A. Mordvintsev, A. Zhmoginov, and M. Vladymyrov. Transformers learn in-context by gradient descent. In *International Conference on Machine Learning*, 2022. URL https://api.semanticscholar.org/CorpusID:254685643.

N. Wandel, M. Weinmann, and R. Klein. Learning incompressible fluid dynamics from scratch–towards fast, differentiable fluid models that generalize. *arXiv preprint arXiv:2006.08762*, 2020.

R. Wang, K. Kashinath, M. Mustafa, A. Albert, and R. Yu. Towards physics-informed deep learning for turbulent flow prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1457–1466, 2020a.

R. Wang, R. Walters, and R. Yu. Incorporating symmetry into deep dynamics models for improved generalization. *ArXiv*, abs/2002.03061, 2020b. URL https://api.semanticscholar.org/CorpusID:211068821.

J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

T. Wolf. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

H. Wu, H. Luo, H. Wang, J. Wang, and M. Long. Transolver: A Fast Transformer Solver for PDEs on General Geometries. In *International Conference on Machine Learning*, 2024. URL http://arxiv.org/abs/2402.02366.

Y. Wu and K. He. Group normalization. *International Journal of Computer Vision*, 128:742 – 755, 2018. URL https://api.semanticscholar.org/CorpusID:4076251.

S. M. Xie, A. Raghunathan, P. Liang, and T. Ma. An explanation of in-context learning as implicit bayesian inference. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=RdJVFCHjUMI.

J. Xu, S. Liu, A. Vahdat, W. Byeon, X. Wang, and S. De Mello. Open-vocabulary panoptic segmentation with text-to-image diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2955–2966, 2023.

L. Yang, S. Liu, T. Meng, and S. J. Osher. In-context operator learning with data prompts for differential equation problems. *Proceedings of the National Academy of Sciences*, 120(39):e2310142120, 2023.

Y. Yin, V. L. Guen, J. Dona, E. de Bézenac, I. Ayed, N. Thome, and P. Gallinari. Augmenting physical models with deep networks for complex dynamics forecasting*. *Journal of Statistical Mechanics: Theory and Experiment*, 2021, 2021. doi: 10.1088/1742-5468/ac3ae5.

Y. Yin, I. Ayed, E. de Bézenac, N. Baskiotis, and P. Gallinari. Leads: Learning dynamical systems that generalize across environments. *Neural Information Processing Systems*, 2022a.

Y. Yin, M. Kirchmeyer, J.-Y. Franceschi, A. Rakotomamonjy, and P. Gallinari. Continuous pde dynamics forecasting with implicit neural representations. *International Conference on Learning Representations*, 9 2022b. URL http://arxiv.org/abs/2209.14855.

Y. Yin, M. Kirchmeyer, J.-Y. Franceschi, A. Rakotomamonjy, and P. Gallinari. Continuous pde dynamics forecasting with implicit neural representations. In *International Conference on Learning Representations, ICLR*, 2023.

L. Yu, Y. Cheng, K. Sohn, J. Lezama, H. Zhang, H. Chang, A. G. Hauptmann, M.-H. Yang, Y. Hao, I. Essa, et al. Magvit: Masked generative video transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10459–10469, 2023a.

L. Yu, J. Lezama, N. B. Gundavarapu, L. Versari, K. Sohn, D. Minnen, Y. Cheng, A. Gupta, X. Gu, A. G. Hauptmann, et al. Language model beats diffusion–tokenizer is key to visual generation. *International Conference on Learning Representations*, 2023b.

B. Zhang and R. Sennrich. Root mean square layer normalization. *ArXiv*, abs/1910.07467, 2019. URL https://api.semanticscholar.org/CorpusID:113405151.

C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. *ArXiv*, abs/1611.03530, 2016. URL https://api.semanticscholar.org/CorpusID:6212000.

X. Zhang, J. Helwig, Y.-C. Lin, Y. Xie, C. Fu, S. Wojtowytsch, and S. Ji. Sinenet: Learning temporal dynamics in time-dependent partial differential equations. *ArXiv*, abs/2403.19507, 2024. URL https://api.semanticscholar.org/CorpusID:268733007.

A. Zhou and A. B. Farimani. Masked autoencoders are pde learners. *arXiv preprint arXiv:2403.17728*, 2024.

L. Zintgraf, K. Shiarli, V. Kurin, K. Hofmann, and S. Whiteson. Fast context adaptation via meta-learning. In *International Conference on Machine Learning*, pages 7693–7702. PMLR, 2019a.

L. Zintgraf, K. Shiarlis, V. Kurin, K. Hofmann, and S. Whiteson. Fast context adaptation via meta-learning. *36th International Conference on Machine Learning, ICML 2019*, 2019-June, 2019b.

Appendix A

Appendix of Chapter 5

A.1 Dataset Details

A.1.1 Initial Value Problem

We use the datasets from Pfaff et al. (2021), and take the first and last frames of each trajectory as the input and output data for the initial value problem.

Cylinder The dataset includes computational fluid dynamics (CFD) simulations of the flow around a cylinder, governed by the incompressible Navier-Stokes equation. These simulations were generated using COMSOL software, employing an irregular 2D-triangular mesh. The trajectory consists of 600 timestamps, with a time interval of $\Delta t = 0.01s$ between each timestamp.

Airfoil The dataset contains CFD simulations of the flow around an airfoil, following the compressible Navier-Stokes equation. These simulations were conducted using SU2 software, using an irregular 2D-triangular mesh. The trajectory encompasses 600 timestamps, with a time interval of $\Delta t = 0.008s$ between each timestamp.

A.1.2 Dynamics Modeling

2D-Navier-Stokes (*Navier-Stokes*) We consider the 2D Navier-Stokes equation as presented in Li et al. (2021); Yin et al. (2022b). This equation models the dynamics of an incompressible fluid on a rectangular domain $\Omega = [-1, 1]^2$. The PDE writes as :

$$\frac{\partial w(x,t)}{\partial t} = -u(x,t)\nabla w(x,t) + \nu \Delta w(x,t) + f, x \in [-1,1]^2, t \in [0,T] \tag{A.1}$$

$$w(x,t) = \nabla \times u(x,t), x \in [-1,1]^2, t \in [0,T] \tag{A.2}$$

$$\nabla u(x,t) = 0, x \in [-1,1]^2, t \in [0,T] \tag{A.3}$$

where u is the velocity, w the vorticity. ν is the fluid viscosity, and f is the forcing term, given by:

$$f(x_1, x_2) = 0.1\left(\sin(2\pi(x_1 + x_2)) + \cos(2\pi(x_1 + x_2))\right), \forall x \in \Omega \tag{A.4}$$

For this problem, we consider periodic boundary conditions.

By sampling initial conditions as in Li et al. (2021), we generated different trajectories on a 256×256 regular spatial grid and with a time resolution $\delta t = 1$. We retain the trajectory starting from the 20th timestep so that the dynamics is sufficiently expressed. The final trajectories contains 40 snapshots at time $t = 20, 21, \cdots, 59$. As explained in Section 5.4, we divide these long trajectories into 2 parts : the 20 first frames are used during the training phase and are denoted as *In-t* throughout this paper. The 20 last timesteps are reserved for evaluating the extrapolation capabilities of the models and are the *Out-t* part of the trajectories. In total, we collected 256 trajectories for training, and 16 for evaluation.

3D-Spherical Shallow-Water (*Shallow-Water*). We consider the shallow-water equation on a sphere describing the movements of the Earth's atmosphere:

$$\frac{\mathrm{d}u}{\mathrm{d}t} = -f \cdot k \times u - g\nabla h + \nu\Delta u \tag{A.5}$$

$$\frac{\mathrm{d}h}{\mathrm{d}t} = -h\nabla \cdot u + \nu\Delta h \tag{A.6}$$

where $\frac{\mathrm{d}}{\mathrm{d}t}$ is the material derivative, k is the unit vector orthogonal to the spherical surface, u is the velocity field tangent to the surface of the sphere, which can be transformed into the vorticity $w = \nabla \times u$, h is the height of the sphere. We generate the data with the *Dedalus* software (Burns et al., 2020), following the setting described in Yin et al. (2022b), where a symmetric phenomena can be seen for both northern and southern hemisphere. The initial zonal velocity u_0 contains two non-null symmetric bands in the both hemispheres, which are parallel to the circles of latitude. At each latitude and longitude $\phi, \theta \in [-\frac{\pi}{2}, \frac{\pi}{2}] \times [-\pi, \pi]$:

$$u_0(\phi, \theta) = \begin{cases} \left(\frac{u_{max}}{e_n} \exp\left(\frac{1}{(\phi-\phi_0)(\phi-\phi_1)}\right), 0\right) & \text{if } \phi \in (\phi_0, \phi_1), \\ \left(\frac{u_{max}}{e_n} \exp\left(\frac{1}{(\phi+\phi_0)(\phi+\phi_1)}\right), 0\right) & \text{if } \phi \in (-\phi_1, -\phi_0), \\ (0, 0) & \text{otherwise.} \end{cases} \tag{A.7}$$

where u_{max} is the maximum velocity, $\phi_0 = \frac{\pi}{7}, \phi_1 = \frac{\pi}{2} - \phi_0$, and $e_n = \exp(-\frac{4}{(\phi_1-\phi_0)^2})$. The water height h_0 is initialized by solving a boundary value conditioned problem

as in Galewsky et al. (2004) which is perturbed by adding h_0' to h_0:

$$h_0'(\phi,\theta) = \hat{h}\cos(\phi)\exp\left(-\left(\frac{\theta}{\alpha}\right)^2\right)\left[\exp\left(-\left(\frac{\phi_2-\phi}{\beta}\right)^2\right)+\exp\left(-\left(\frac{\phi_2+\phi}{\beta}\right)^2\right)\right]. \tag{A.8}$$

where $\phi_2 = \frac{\pi}{4}, \hat{h} = 120\text{m}, \alpha = \frac{1}{3}$ and $\beta = \frac{1}{15}$ are constants defined in Galewsky et al. (2004). We simulated the phenomenon using Dedalus Burns et al. (2020) on a latitude-longitude grid (lat-lon). The original grid size was 128 (lat) $\times$ 256 (lon), which we downsampled to obtain grids of size 64×128. To generate trajectories, we sampled u_{max} from a uniform distribution $\mathcal{U}(60, 80)$. Snapshots were captured every hour over a duration of 320 hours, resulting in trajectories with 320 timestamps. We created 16 trajectories for the training set and 2 trajectories for the test set. However, since the dynamical phenomena in the initial timestamps were less significant, we only considered the last 160 snapshots. Each long trajectory is then sliced into sub-trajectories of 40 timestamps each. As a result, the training set contains 64 trajectories, while the test set contains 8 trajectories. It is worth noting that the data was also scaled to a reasonable range: the height h was scaled by a factor of 3×10^3, and the vorticity w was scaled by a factor of 2.

A.1.3 Geometric aware inference

We use the datasets provided by Li et al. (2022a) and adopt the original authors' train/test split for our experiments.

Euler's Equation (*Naca-Euler*). We consider the transonic flow over an airfoil, where the governing equation is Euler equation, as follows:

$$\frac{\partial \rho_f}{\partial t} + \nabla\cdot(\rho_f u) = 0, \frac{\partial \rho_f u}{\partial t} + \nabla\cdot(\rho_f u\otimes u + p\mathbb{I}) = 0, \frac{\partial E}{\partial t} + \nabla\cdot((E+p)u) = 0, \tag{A.9}$$

where ρ_f is the fluid density, u is the velocity vector, p is the pressure, and E is the total energy. The viscous effect is ignored. The far-field boundary condition is $\rho_\infty = 1$, $p_\infty = 1.0$, $M_\infty = 0.8$, $AoA = 0$, where M_∞ is the Mach number and AoA is the angle of attack. At the airfoil, a no-penetration condition is imposed. The shape parameterization of the airfoil follows the design element approach. The initial NACA-0012 shape is mapped onto a "cubic" design element with 8 control nodes, and the initial shape is morphed to a different one following the displacement field of the control nodes of the design element. The displacements of control nodes are restricted to the vertical direction only, with prior $d \sim \mathcal{U}[-0.05, 0.05]$.

We have access to 1000 training data and 200 test data, generated with a second-order implicit finite volume solver. The C-grid mesh with about (200×50) quadrilateral elements is used, and the mesh is adapted near the airfoil but not the shock.

The mesh point locations and Mach number on these mesh points are used as input and output data.

Hyper-elastic material (*Elasticity*). The governing equation of a solid body can be written as

$$\rho_s \frac{\partial^2 u}{\partial t^2} + \nabla \cdot \sigma = 0$$

where ρ_s is the mass density, u is the displacement vector, and σ is the stress tensor. Constitutive models, which relate the strain tensor ε to the stress tensor, are required to close the system. We consider the unit cell problem $\Omega = [0, 1] \times [0, 1]$ with an arbitrary shape void at the center, which is depicted in Figure 2(a). The prior of the void radius is $r = 0.2 + 0.2$ with $\tilde{r} \sim \mathcal{N}(0, 42(-\nabla + 32)^{-1})$, $1 + \exp(\tilde{r})$, which embeds the constraint $0.2 \leq r \leq 0.4$. The unit cell is clamped on the bottom edges and tension traction $t = [0, 100]$ is applied on the top edge. The material is the incompressible Rivlin-Saunders material with energy density function parameters $C_1 = 1.863 \times 10^5$ and $C_1 = 9.79 \times 10^3$. The data was generated with a finite element solver with about 100 quadratic quadrilateral elements. The inputs a are given as point clouds with a size around 1000. The target output is stress.

Navier-Stokes Equation (*Pipe*). We consider the incompressible flow in a pipe, where the governing equation is the incompressible Navier-Stokes equation, as following,

$$\frac{\partial v}{\partial t} + (v \cdot \nabla)v = -\nabla p + \mu \nabla^2 v, \quad \nabla \cdot v = 0$$

where v is the velocity vector, p is the pressure, and $\mu = 0.005$ is the viscosity. The parabolic velocity profile with maximum velocity $v = [1, 0]$ is imposed at the inlet. A free boundary condition is imposed at the outlet, and a no-slip boundary condition is imposed at the pipe surface. The pipe has a length of 10 and width of 1. The centerline of the pipe is parameterized by 4 piecewise cubic polynomials, which are determined by the vertical positions and slopes on 5 spatially uniform control nodes. The vertical position at these control nodes obeys $d \sim \mathcal{U}[-2, 2]$, and the slope at these control nodes obeys $d \sim \mathcal{U}[-1, 1]$.

We have access to 1000 training data and 200 test data, generated with an implicit finite element solver using about 4000 Taylor-Hood Q2-Q1 mixed elements. The mesh point locations (129×129) and horizontal velocity on these mesh points are used as input and output data.

A.2 Implementation Details

We implemented all experiments with PyTorch (Paszke et al., 2019b). We estimate the computation time needed for development and the different experiments

to approximately 400 days.

A.2.1 CORAL

A.2.1.1 Architecture Details

SIREN initialization. We use for SIREN the same initialization scheme as in Sitzmann et al. (2020c), i.e., sampling the weights of the first layer according to a uniform distribution $\mathcal{U}(-1/d, 1/d)$ and the next layers according to $\mathcal{U}(-\frac{1}{w_0}\sqrt{\frac{6}{d_{in}}}, \frac{1}{w_0}\sqrt{\frac{6}{d_{in}}})$. We use the default PyTorch initialization for the hypernetwork.

Decode with shift-modulated SIREN. Initially, we attempted to modulate both the scale and shift of the activation, following the approach described in Perez et al. (2018b). However, we did not observe any performance improvement by employing both modulations simultaneously. Consequently, we decided to focus solely on shift modulations, as it led to a more stable training process and reduced the size of the modulation space by half. We provide an overview of the decoder with the shift-modulated SIREN in Figure A.1.

Encode with *auto-decoder*. We provide a schematic view of the input encoder in Figure A.2. The *auto-decoding* process starts from a code $z_a = 0$ and performs K steps of gradient descent over this latent code to minimize the reconstruction loss.

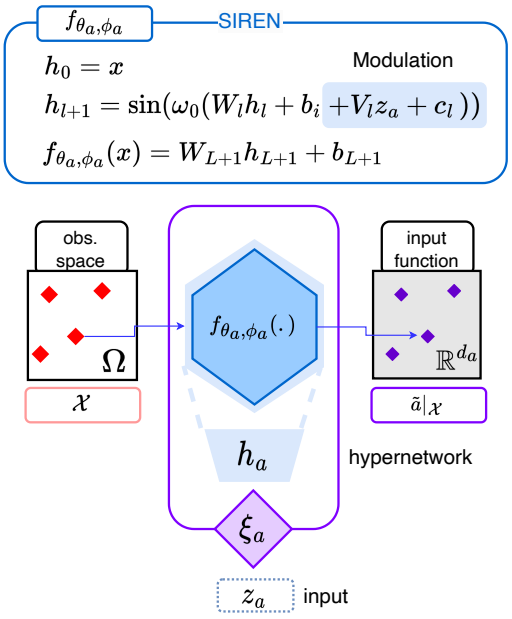
Process with MLP. We use an MLP with skip connections and Swish activation functions. Its forward function writes $g_\psi(z) = \text{Block}_k \circ ... \circ \text{Block}_1(z)$, where Block is a two-layer MLP with skip connections:

$$\text{Block}(z) = z + \sigma(\mathbf{W}_2 \cdot \sigma(\mathbf{W}_1 \cdot z + \mathbf{b}_1) + \mathbf{b}_2) \tag{A.10}$$

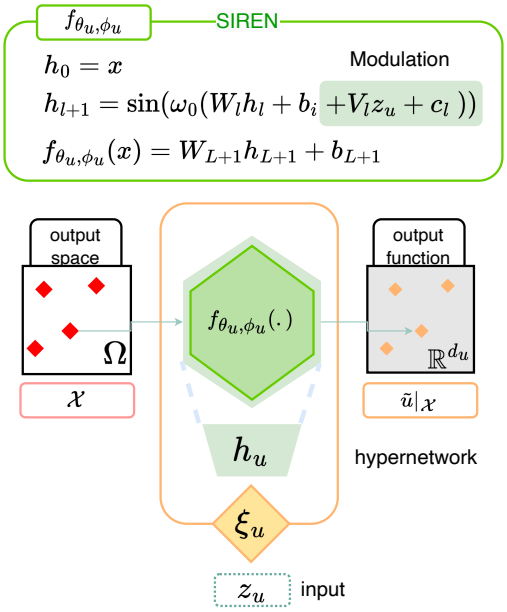
In Equation (A.10), σ denotes the feature-wise Swish activation. We use the version with learnable parameter β; $\sigma(z) = z \cdot \text{sigmoid}(\beta z)$.

A.2.1.2 Training Details

The training is done in two steps. First, we train the modulated INRs to represent the data. We show the details with the pseudo-code in Algorithms 4 and 5. α is the inner-loop learning rate while λ is the outer loop learning rate, which adjusts the weights of the INR and hypernetwork. Then, once the INRs have been fitted, we obtain the latent representations of the training data, and use these latent codes to train the forecast model g_ψ (See Algorithm 6). We note λ_ψ the learning rate of g_ψ.



(a) The hypernetwork h_a maps the input code z_a to the modulations ϕ_a. The modulations shift the activations at each layer of the SIREN.



(b) The hypernetwork h_u maps the input code z_u to the modulations ϕ_u. The modulations shift the activations at each layer of the SIREN.

Figure A.1: Architecture of the input and output decoders ξ_a, ξ_u. They can be queried on any coordinate $x \in \Omega$. We use the same notation for both, even though the parameters are different.

Z-score normalization. As the data is encoded using only a few steps of gradients, the resulting standard deviation of the codes is very small, falling within the range of [1e-3, 5e-2]. However, these "raw" latent representations are not suitable as-is for further processing. To address this, we normalize the codes by subtracting the mean and dividing by the standard deviation, yielding the normalized code: $z_{\text{norm}} = \frac{z-\text{mean}}{\text{std}}$. Depending on the task, we employ slightly different types of normalization:

1. Initial value problem: • *Cylinder*: We normalize the inputs and outputs code with the same mean and standard deviation. We compute the statistics feature-wise, across the inputs and outputs. • *Airfoil*: We normalize the inputs and outputs code with their respective mean and standard deviation. The statistics are real values.

2. Dynamics modeling: We normalize the codes with the same mean and standard deviation. The statistics are computed feature-wise, over all training trajectories and all available timestamps (i.e. over *In-t*).

3. Geometry-aware inference: We normalize the input codes only, with feature-wise statistics.

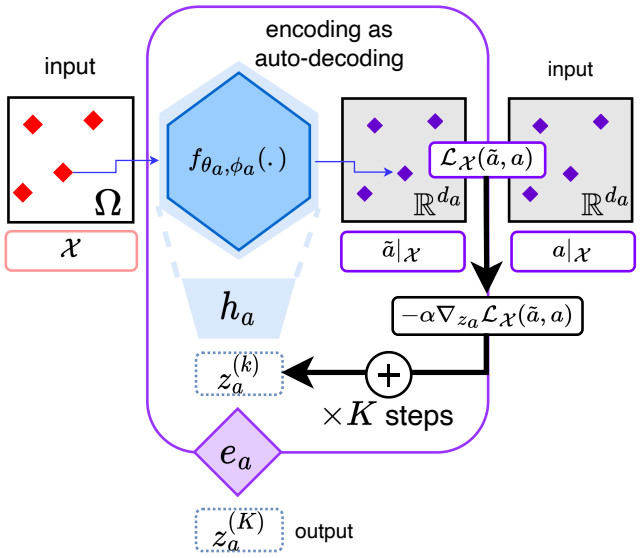


Figure A.2: Starting from a code $z_a^{(0)} = 0$, the input encoder e_a performs K inner steps of gradient descent over z_a to minimize the reconstruction loss $\mathcal{L}_{\mathcal{X}}(\tilde{a}, a)$ and outputs the resulting code $z_a^{(K)}$ of this optimization process. During training, we accumulate the gradients of this encoding phase and back-propagate through the K inner-steps to update the parameters θ_a and w_a. At inference, we encode new inputs with the same number of steps K and the same learning rate α, unless stated otherwise. The output encoder works in the same way during training, and is not used at inference.

A.2.1.3 Inference Details

We present the inference procedure in Algorithm 7. It is important to note that the input and output INRs, f_{θ_a} and f_{θ_u}, respectively, accept the "raw" codes as inputs, whereas the processor expects a normalized latent code. Therefore, after the encoding steps, we normalize the input code. Additionally, we may need to denormalize the code immediately after the processing stage. It is worth mentioning that we maintain the same number of inner steps as used during training, which is 3 for all tasks.

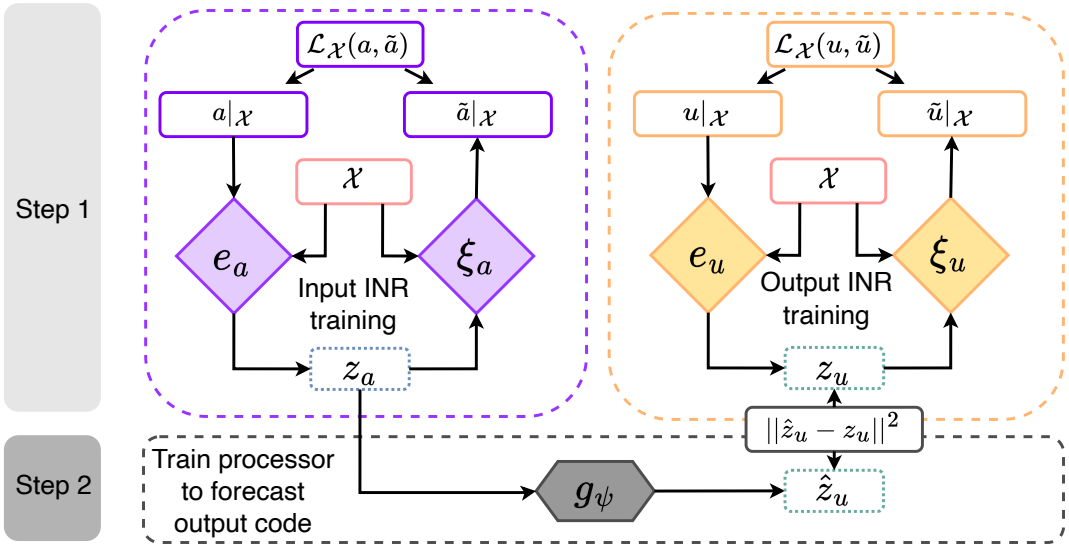


Figure A.3: Proposed training for CORAL. (1) We first learn to represent the data with the input and output INRs. (2) Once the INRs are trained, we obtain the latent representations and fix the pairs of input and output codes (z_{a_i}, z_{u_i}). We then train the processor to minimize the distance between the processed code $g_\psi(z_{a_i})$ and the output code z_{u_i}.

Algorithm 4: Training of the input INR

while no convergence **do**
  Sample batch $\mathcal{B}$ of data $(a_i)_{i\in\mathcal{B}}$;
  Set codes to zero $z_{a_i} \leftarrow 0, \forall i \in \mathcal{B}$;
  for $i \in \mathcal{B}$ and step $\in \{1, ..., K_a\}$ **do**
    $z_{a_i} \leftarrow z_{a_i} - \alpha_a \nabla_{z_{a_i}} \mathcal{L}_{\mathcal{X}_i}(f_{\theta_a, h_a(z_{a_i})}, a_i)$; `// input encoding inner step`
  end
  `/* outer loop update */`
  $\theta_a \leftarrow \theta_a - \lambda \frac{1}{|\mathcal{B}|} \sum_{i\in\mathcal{B}} \nabla_{\theta_a} \mathcal{L}_{\mathcal{X}_i}(f_{\theta_a, h_a(z_{a_i})}, a_i)$;
  $w_a \leftarrow w_a - \lambda \frac{1}{|\mathcal{B}|} \sum_{i\in\mathcal{B}} \nabla_{w_a} \mathcal{L}_{\mathcal{X}_i}(f_{\theta_a, h_a(z_{a_i})}, a_i)$
end

Algorithm 5: Training of the output INR

while no convergence **do**
  Sample batch $\mathcal{B}$ of data $(a_i, u_i)_{i\in\mathcal{B}}$;
  Set codes to zero $z_{u_i} \leftarrow 0, \forall i \in \mathcal{B}$;
  for $i \in \mathcal{B}$ and step $\in \{1, ..., K_u\}$ **do**
    $z_{u_i} \leftarrow z_{u_i} - \alpha_u \nabla_{z_{u_i}} \mathcal{L}_{\mathcal{X}_i}(f_{\theta_u, h_u(z_{u_i})}, u_i)$; `// output encoding inner step`
  end
  `/* outer loop update */`
  $\theta_u \leftarrow \theta_u - \lambda \frac{1}{|\mathcal{B}|} \sum_{i\in\mathcal{B}} \nabla_{\theta_u} \mathcal{L}_{\mathcal{X}_i}(f_{\theta_u, h_u(z_{u_i})}, u_i)$;
  $w_u \leftarrow w_u - \lambda \frac{1}{|\mathcal{B}|} \sum_{i\in\mathcal{B}} \nabla_{w_u} \mathcal{L}_{\mathcal{X}_i}(f_{\theta_u, h_u(z_{u_i})}, u_i)$
end

Algorithm 6: Training of the processor

while no convergence **do**
  Sample batch $\mathcal{B}$ of codes $(z_{a_i}, z_{u_i})_{i\in\mathcal{B}}$;
  `/* processor update */`
  $\psi \leftarrow \psi - \lambda_\psi \frac{1}{|\mathcal{B}|} \sum_{i\in\mathcal{B}} \nabla_\psi \mathcal{L}(g_\psi(z_{a_i}), z_{u_i})$;
end

Algorithm 7: CORAL Inference, given a function a

Set code to zero $z_a \leftarrow 0$;
for step $\in \{1, ..., K_a\}$ **do**
  $z_a \leftarrow z_a - \alpha_a \nabla_{z_a} \mathcal{L}_\mathcal{X}(f_{\theta_a, h_a(z_a)}, a)$; `// input encoding inner step`
end
$\hat{z}_u = g_\psi(z_a)$; `// process latent code`
$\hat{u} = f_{\theta_u, h_u(\hat{z}_u)}$; `// decode output function`

A.2.1.4 Choice of Hyperparameters

We recall that d_z denotes the size of the code, w_0 is a hyperparameter that controls the frequency bandwith of the SIREN network, λ is the outer-loop learning rate (on $f_{\theta,\phi}$ and h_w), α is the inner-loop learning rate, K is the number of inner steps used during training and encoding steps at test time, λ_ψ is the learning rate of the MLP or NODE. In some experiments we learn the inner-loop learning rate α, as in Li et al. (2017b). In such case, the meta-α learning rate is an additional parameter that controls how fast we move α from its initial value during training. When not mentioned we simply report α in the tables below, and otherwise we report the initial learning rate and this meta-learning-rate.

We use the Adam optimizer during both steps of the training. For the training of the Inference / Dynamics model, we use a learning rate scheduler which reduces the learning rate when the loss has stopped improving. The threshold is set to 0.01 in the default relative threshold model in PyTorch, with a patience of 250 epochs w.r.t. the train loss. The minimum learning rate is 1e-5.

Initial Value Problem We provide the list of hyperparameters used for the experiments on *Cylinder* and *Airfoil* in Table A.1.

Dynamics Modeling Table A.2 summarizes the hyperparameters used in our experiments for dynamics modeling on datasets *Navier-Stokes* and *Shallow-Water* (Table 5.2).

Furthermore, to facilitate the training of the dynamics within the NODE, we employ Scheduled Sampling, following the approach described in Bengio et al. (2015). At each timestep, there is a probability of ϵ% for the integration of the dynamics

Table A.1: CORAL hyper-parameters for IVP/ Geometry-aware inference

	Hyper-parameter	*Cylinder*	*Airfoil*	*NACA-Euler*	*Elasticity*	*Pipe*
f_{θ_a,ϕ_a} / f_{θ_u,ϕ_u}	d_z	128	128	128	128	128
	depth	4	5	4	4	5
	width	256	256	256	256	128
	ω_0	30	30 / 50	5 / 15	10 / 15	5 / 10
SIREN Optimization	batch size	32	16	32	64	16
	epochs	2000	1500	5000	5000	5000
	λ	5e-6	5e-6	1e-4	1e-4	5e-5
	α	1e-2	1e-2	1e-2	1e-2	1e-2
	meta-α learning rate	0	5e-6	1e-4	1e-4	5e-5
	K_a / K_u	3	3	3	3	3
g_ψ	depth	3	3	3	3	3
	width	64	64	64	64	128
	activation	Swish	Swish	Swish	Swish	Swish
Inference Optimization	batch size	32	16	64	64	64
	epochs	100	100	10000	10000	10000
	λ_ψ	1e-3	1e-3	1e-3	1e-3	1e-3
	Scheduler decay	0	0	0.9	0.9	0.9

Table A.2: CORAL hyper-parameters for dynamics modeling

	Hyper-parameter	*Navier-Stokes*	*Shallow-Water*
INR	d_z	128	256
	depth	4	6
	width	128	256
	ω_0	10	10
INR Optimization	batch size	64	16
	epochs	10, 000	10, 000
	λ	5e-6	5e-6
	α	1e-2	1e-2
	K	3	3
NODE	depth	3	3
	width	512	512
	activation	Swish	Swish
	solver	RK4	RK4
Dynamics Optimization	batch size	32	16
	epochs	10, 000	10, 000
	λ_ψ	1e-3	1e-3
	Scheduler decay	0.75	0.75

through the ODE solver to be restarted using the training snapshots. This probability gradually decreases during the training process. Initially, we set $\epsilon_{\text{init}} = 0.99$, and every 10 epochs, we multiply it by 0.99. Consequently, by the end of the training procedure, the entire trajectory is computed with the initial condition.

Geometry-aware inference We provide the list of hyperparameters used for the experiments on *NACA-Euler*, *Elasticity*, and *Pipe* in Table A.1.

A.2.2 Baseline Implementation

We detail in this section the architecture and hyperparameters used for the training of the baselines presented in Section 5.4.

Initial Value Problem We use the following baselines for the Initial Value Problem task.

- **NodeMLP**. We use a ReLU-MLP with 3 layers and 512 neurons. We train it for 10000 epochs. We use a learning rate of 1e-3 and a batch size of 64.
- **GraphSAGE**. We use the implementation from torch-geometric (Fey and Lenssen, 2019), with 6 layers of 64 neurons. We use ReLU activation. We train the model for 400 epochs for *Airfoil* and 4,000 epochs for *Cylinder*. We build the graph using the 16 closest nodes. We use a learning rate of 1e-3 and a batch size of 64.
- **MP-PDE**: We implement MP-PDE as a 1-step solver, where the time-bundling and pushforward trick do not apply. We use 6 message-passing blocks and 64 hidden features. We build the graph with the 16 closest nodes. We use a learning rate of 1e-3 and a batch size of 16. We train for 500 epochs on *Airfoil* and 1000 epochs on *Cylinder*.

Dynamics Modeling All our baselines are implemented in an auto-regressive (AR) manner to perform forecasting.

- **DeepONet**: We use a DeepONet in which both Branch Net and Trunk Net are 4-layers MLP with 100 neurons. The model is trained for $10,000$ epochs with a learning rate of 1e-5. To complete the upsampling studies, we used a modified DeepONet forward which computes as follows: (1) Firstly, we compute an AR pass on the training grid to obtain a prediction of the complete trajectory with the model on the training grid. (2) We use these prediction as input of the branch net for a second pass on the up-sampling grid to obtain the final prediction on the new grid.

- **FNO**: FNO is trained for $2,000$ epochs with a learning rate of 1e-3. We used 12 modes and a width of 32 and 4 Fourier layers. We also use a step scheduler every 100 epochs with a decay of 0.5.
- **MP-PDE**: We implement MP-PDE with a time window of 1 so that is becomes AR. The MP-PDE solver is composed of a 6 message-passing blocks with 128 hidden features. To build the graphs, we limit the number of neighbors to 8. The optimization was performed on $10,000$ epochs with a learning rate of 1e-3 and a step scheduler every 2000 epochs until 10000. We decay the learning rate of 0.4 with weight decay 1e-8.
- **DINo**: DINo uses MFN model with respectively width and depth of 64 and 3 for Navier-Stokes (NS), and 256 and 6 for Shallow-Water (SW). The encoder proceeds to 300 (NS) or 500 (SW) steps to optimize the codes whose size is set to 100 (NS) or 200 (SW). The dynamic is solved with a NODE that uses 4-layers MLP and a hidden dimension of 512 (NS) or 800 (SW). This model is trained for $10,000$ epochs with a learning rate of 5e-3. We use the same scheduled sampling as for the CORAL training (see Appendix A.2.1.4).

Geometry-aware inference Except on *Pipe*, the numbers for **FactorizedFNO** are taken from Tran et al. (2023). In the latter we take the 12-layer version which has a comparable model size. We train the 12-layer FactorizedFNO on *Pipe* with AdamW for 200 epochs with modes (32, 16), a width of 64, a learning rate of 1e-3 and a weight decay of 1e-4. We implemented the baselines **GeoFNO**, **FNO**, **UNet** according to the code provided in Li et al. (2022a)

A.3 Supplementary Results for Dynamics Modeling

A.3.1 Robustness to Resolution Changes

We present in Tables A.3 and A.4 the up-sampling capabilities of CORAL and relevant baselines both *In-t* and *Out-t*, respectively for *Navier-Stokes* and *Shallow-Water*.

These tables show that CORAL remains competitive and robust on up-sampled inputs. Other baselines can also predict on denser grids, except for MP-PDE, which over-fitted the training grid.

Table A.3: Up-sampling capabilities - Test results on *Navier-Stokes* dataset. Metrics in MSE.

$\mathcal{X}_{tr} \downarrow$	dataset $\rightarrow$	*Navier-Stokes*							
	$\mathcal{X}_{tr} \rightarrow$	64×64							
	$\mathcal{X}_{te} \rightarrow$	$\mathcal{X}_{tr}$		64×64		128×128		256×256	
		In-t	*Out-t*	*In-t*	*Out-t*	*In-t*	*Out-t*	*In-t*	*Out-t*
$\pi_{tr} = 100\%$ regular grid	DeepONet	1.47e-2	7.90e-2	1.47e-2	7.90e-2	1.82e-1	7.90e-2	1.82e-2	7.90e-2
	FNO	7.97e-3	1.77e-2	7.97e-3	1.77e-2	8.04e-3	1.80e-2	1.81e-2	7.90e-2
	MP-PDE	5.98e-4	2.80e-3	5.98e-4	2.80e-3	2.36e-2	4.61e-2	4.26e-2	9.77e-2
	DINo	1.25e-3	1.13e-2	1.25e-3	1.13e-2	1.25e-3	1.13e-2	1.26e-3	1.13e-2
	CORAL	**2.02e-4**	**1.07e-3**	**2.02e-4**	**1.07e-3**	**2.08e-4**	**1.06e-3**	**2.19e-4**	**1.07e-3**
$\pi_{tr} = 20\%$ irregular grid	DeepONet	8.35e-1	7.74e-1	8.28e-1	7.74e-1	8.32e-1	7.74e-1	8.28e-1	7.73e-1
	MP-PDE	2.36e-2	1.11e-1	7.42e-2	2.13e-1	1.18e-1	2.95e-1	1.37e-1	3.39e-1
	DINo	**1.30e-3**	9.58e-3	**1.30e-3**	9.59e-3	**1.31e-3**	9.63e-3	**1.32-3**	9.65e-3
	CORAL	1.73e-3	**5.61e-3**	1.55e-3	**4.34e-3**	1.61e-3	**4.38e-3**	1.65e-3	**4.41e-3**
$\pi_{tr} = 5\%$ irregular grid	DeepONet	7.12e-1	7.16e-1	7.22e-1	7.26e-1	7.24e-1	7.28e-1	7.26e-1	7.30e-1
	MP-PDE	1.25e-1	2.92e-1	4.83e-1	1.08	6.11e-1	1.07	6.49e-1	1.08
	DINo	8.21e-2	1.03e-1	7.73e-2	7.49e-2	7.87e-2	7.63e-2	7.96e-2	7.73e-2
	CORAL	**1.56e-2**	**3.65e-2**	**4.19e-3**	**1.12e-2**	**4.30e-3**	**1.14e-2**	**4.37e-3**	**1.14e-2**

A.3.2 Learning a Dynamics on Different Grids

To extend our work, we propose to study how robust is CORAL to changes in grids. In our classical setting, we keep the same grid for all trajectories in the training set and evaluate it on a new grid for the test set. Instead, here, both in train and test sets, each trajectory i has its own grid $\mathcal{X}_i$. Thus, we evaluate CORAL's capability to generalize to grids. We present the results in Table A.5. Overall, coordinate-based methods generalize better over grids compared to operator based and discrete methods like DeepONet and MP-PDE which show better or equivalent performance when trained only on one grid. CORAL's performance is increased when trained on different grids; one possible reason is that CORAL overfits the training grid used for all trajectories in our classical setting.

A.3.3 Training Time

In Table A.6, we present the training time for CORAL and different baselines for comparison. Since, the training of CORAL is separated in 2 steps, we show in line "INR" the training time for INR fitting and in line "Process" the second step to train the forecast model. We see that the longest part of the training procedure in CORAL is the fitting of the INR. MP-PDE is the slowest baseline to train, due to the KNN graph creation. DeepONet and FNO are the fastest baselines to train because they only need a forward pass.

Table A.4: Up-sampling capabilities - Test results on *Shallow-Water* dataset. Metrics in MSE.

$\mathcal{X}_{tr} \downarrow$	dataset → / $\mathcal{X}_{tr}$ → / $\mathcal{X}_{te}$ →	*Shallow-Water* 64 × 128							
		$\mathcal{X}_{tr}$		32 × 64		64 × 128		128 × 256	
		In-t	*Out-t*	*In-t*	*Out-t*	*In-t*	*Out-t*	*In-t*	*Out-t*
$\pi_{tr} = 100\%$ regular grid	DeepONet	7.07e-3	9.02e-3	1.18e-2	1.66e-2	7.07e-3	9.02e-3	1.18e-2	1.66e-2
	FNO	6.75e-5	**1.49e-4**	7.54e-5	**1.78e-4**	6.75e-5	**1.49e-4**	6.91e-5	**1.52e-4**
	MP-PDE	2.66e-5	4.35e-4	4.80e-2	1.42e-2	2.66e-5	4.35e-4	4.73e-3	1.73e-3
	DINo	4.12e-5	2.91e-3	5.77e-5	2.55e-3	4.12e-5	2.91e-3	6.04e-5	2.58e-3
	CORAL	**3.52e-6**	4.99e-4	**1.86e-5**	5.32e-4	**3.52e-6**	4.99e-4	**4.96e-6**	4.99e-4
irregular grid $\pi_{tr} = 20\%$	DeepONet	1.08e-2	1.10e-2	2.49e-2	3.25e-2	2.49e-2	3.25e-2	2.49e-2	3.22e-2
	MP-PDE	4.54e-3	1.48e-2	4.08e-3	1.30e-2	5.46e-3	1.74e-2	4.98e-3	1.43e-2
	DINo	2.32e-3	5.18e-3	2.22e-3	4.80e-3	2.16e-3	4.64e-3	2.16e-3	4.64e-3
	CORAL	**1.36e-3**	**2.17e-3**	**1.24e-3**	**1.95e-3**	**1.21e-3**	**1.95e-3**	**1.21e-3**	**1.95e-3**
irregular grid $\pi_{tr} = 5\%$	DeepONet	1.02e-2	**1.01e-2**	1.57e-2	1.93e-2	1.57e-2	1.93e-2	1.57e-2	1.93e-2
	MP-PDE	**5.36e-3**	1.81e-2	**5.53e-3**	1.80e-2	**4.33e-3**	1.32e-2	**5.48e-3**	1.74e-2
	DINo	1.25e-2	1.51e-2	1.39e-2	1.54e-2	1.39e-2	1.54e-2	1.39e-2	1.54e-2
	CORAL	8.40e-3	1.25e-2	9.27e-3	**1.15e-2**	9.26e-3	**1.16e-2**	9.26e-3	**1.16e-2**

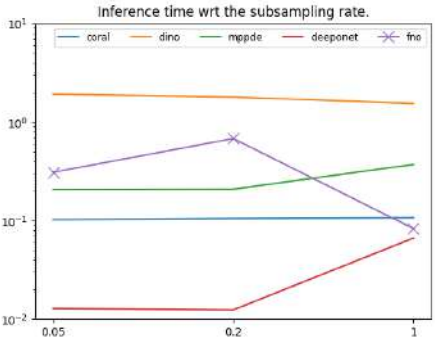
A.3.4 Inference Time

In this section, we evaluate the inference time of CORAL and other baselines w.r.t. the input grid size. We study the impact of the training grid size (different models trained with 5%, 20% and 100% of the grid) (Figure A.4a) and the time needed for a model trained (5%) on a given grid to make computation on finer grid size resolution (evaluation grid size) (Figure A.4b).

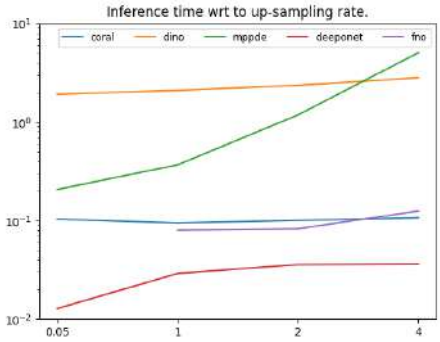
On the graphs presented in Figure A.4, we observe that except for the operator baselines, CORAL is also competitive in terms of inference time. MP-PDE inference time increases strongly when inference grid gets denser. The DINo model, which is the only to propose the same properties as CORAL, is much slower when both inference and training grid size evolve. This difference is mainly explained by the number of steps needed to optimize DINo codes. Indeed, at test time DINO's adaptation requires 100x more optimization steps. MPPDE's computation is slow due to the KNN graph creation and slower message passing. DeepONet and FNO are faster due to forward computation only. CORAL's encoding/decoding is relatively resolution-insensitive and performed in parallel across all sensors. Process operates on a fixed dimension independent of the resolution. FNO is fast due to FFT but cannot be used on irregular grids. For these experiments, we used linear interpolation which slowed the inference time.

Table A.5: Learning dynamics on different grids - Test results in the extrapolation setting. Metrics in MSE.

$\mathcal{X}_{tr} \downarrow \mathcal{X}_{te}$	dataset →	*Navier-Stokes*		*Shallow-Water*	
		In-t	*Out-t*	*In-t*	*Out-t*
$\pi = 20\%$ irregular grid	DeepONet	5.22×10^{-1}	5.00×10^{-1}	1.11×10^{-2}	1.12×10^{-2}
	MP-PDE	6.11×10^{-1}	6.10×10^{-1}	6.80×10^{-3}	1.87×10^{-2}
	DINo	1.30e-3	1.01e-2	4.12e-4	3.05e-3
	CORAL	3.21×10^{-4}	3.03×10^{-3}	1.15×10^{-4}	7.75×10^{-4}
$\pi = 5\%$ irregular grid	DeepONet	4.11×10^{-1}	4.38×10^{-1}	1.11×10^{-2}	1.12×10^{-2}
	MP-PDE	8.15×10^{-1}	1.10×10^{-1}	1.22×10^{-2}	4.29×10^{-2}
	DINo	1.26e-3	1.04e-2	3.89e-3	7.41e-3
	CORAL	9.82×10^{-4}	9.71×10^{-3}	**2.22e-3**	**4.89e-3**



(a) Average inference time (in seconds) of the implemented baselines to unroll a trajectory until $T = 40$ on *Navier-Stokes*. For irregular grids, FNO is performed following linear interpolation.

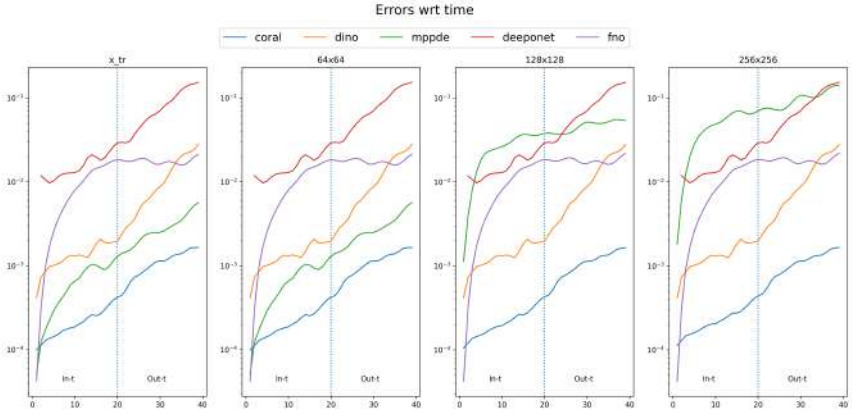


(b) Inference time w.r.t. evaluation grid size (same models).

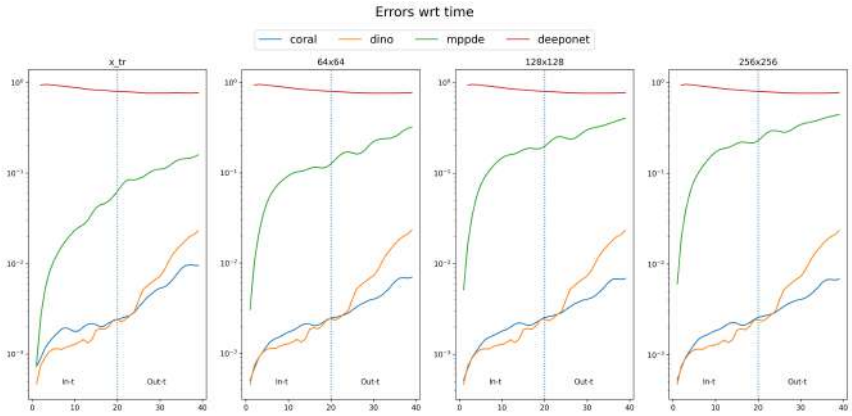
Figure A.4: Inference time analysis of CORAL vs baselines

A.3.5 Propagation of Errors Through Time

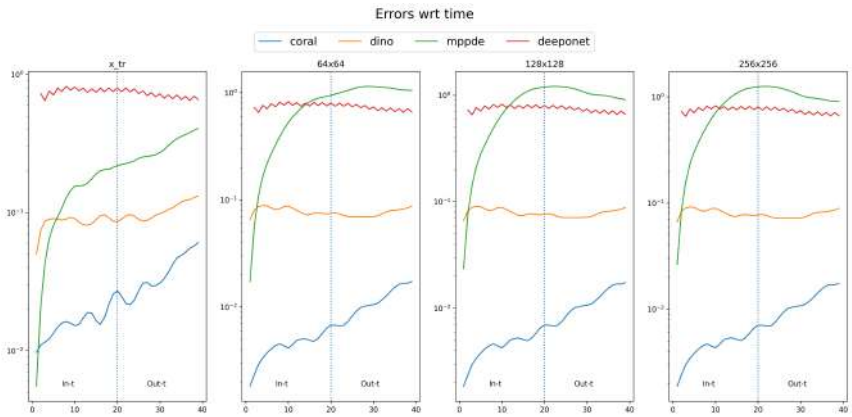
In Figures A.5a to A.5c, we show the evolution of errors as the extrapolation horizon evolves. First, we observe that all baselines propagate error through time, since the trajectories are computed using an auto-regressive approach. Except for the 100%, DeepONet had difficulties to handle the dynamic. It has on all settings the highest error. Then, we observe that for MP-PDE and FNO, the error increases quickly at the beginning of the trajectories. This means that these two models are rapidly propagating error. Finally, both DINo and CORAL have slower increase of the error



(a) Evolution of errors over time and across test samples for a model trained on 100% of the grid.



(b) Evolution of errors over time and across test samples for a model trained on 20% of the grid.



(c) Evolution of errors over time and across test samples for a model trained on 5% of the grid.

Figure A.5: Errors along a given trajectory.

Table A.6: Training time comparison - Expressed in days (d) or hours (h) on several datasets.

Model	*Cylinder*	*Navier-Stokes*	*Shallow-Water*	*Elasticity*	*NACA*
CORAL (INR)	6h	1d	5d	4h	2d
CORAL (Process)	1h	4h	1h	1h	1h
NodeMLP	0.5h	-	-	-	-
GraphSAGE	1d	-	-	-	-
MP-PDE	7h	19h	21h	-	-
DINo	-	8h	2d	-	-
DeepONet	-	6h	5h	-	-
FNO	-	8h	6h	0.5h	0.5h
UNet	-	-	-	0.5h	0.5h
Geo-FNO	-	-	-	1.0h	1.0h
Factorized-FNO	-	-	-	1.0h	1.0h

during *In-t* and *Out-t* periods. However, we clearly see on the graphs that DINo has more difficulties than CORAL to make predictions out-range. Indeed, while CORAL's error augmentation remains constant as long as the time evolves, DINo has a clear increase.

A.3.6 Benchmarking INRs for CORAL

We provide some additional experiments for dynamics modeling with CORAL, but with diffrents INRs: MFN (Fathony et al., 2021b), BACON (Lindell et al., 2022) and FourierFeatures (Tancik et al., 2020b). Experiments have been done on Navier-Stokes on irregular grids sampled from grids of size 128×128. All training trajectories share the same grid and are evaluated on a new grid for test trajectories. Results are reported in Table A.7. Note that we used the same learning hyper-parameters for the baselines than those used for SIREN in CORAL. SIREN seems to produce the best codes for dynamics modeling, both for in-range and out-range prediction.

A.3.7 Impact of 2nd order meta-learning

We provide in Figure A.6 the evolution of the reconstruction error through the training epochs for *Navier-Stokes* with first-order and second-order meta-learning. The first order method is able to correctly train until it falls into an unstable regime for the common parameters. The second order method is much more stable and achieves a 10x reduction in MSE.

Table A.7: CORAL results with different INRs. - Test results in the extrapolation setting on *Navier-Stokes* dataset. Metrics in MSE.

$\mathcal{X}_{tr} \downarrow \mathcal{X}_{te}$	INR	*In-t*	*Out-t*
$\pi = 20\%$ irregular grid	SIREN	**5.76e-4**	**2.57e-3**
	MFN	2.21e-3	5.17e-3
	BACON	2.90e-2	3.32e-2
	FourierFeatures	1.70e-3	5.67e-3
$\pi = 5\%$ irregular grid	SIREN	**1.81e-3**	**4.15e-3**
	MFN	9.97e-1	9.58e-1
	BACON	1.06	8.06e-1
	FourierFeatures	3.60e-1	3.62e-1

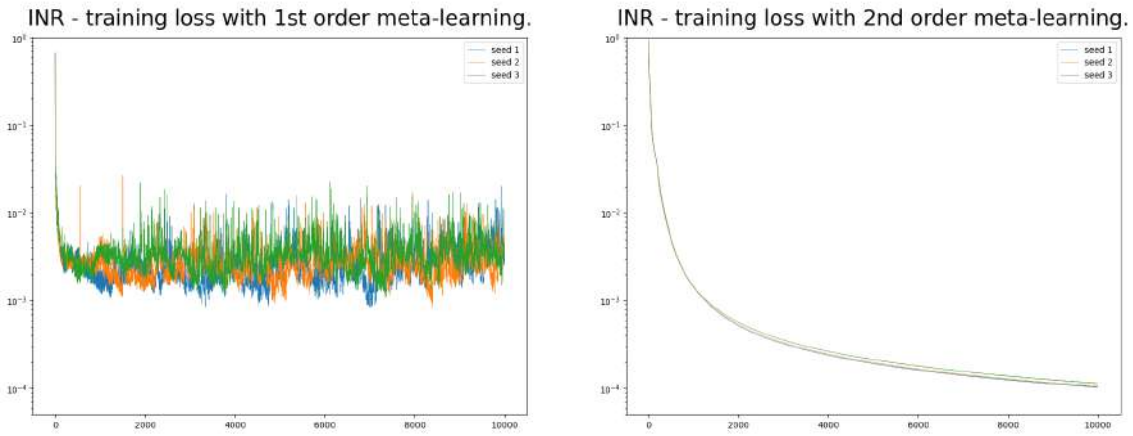


Figure A.6: Training of the modulated INR - Comparison on *Navier-Stokes* over three independent runs of first order and second order meta-learning. We use the same number of inner-steps.

A.3.8 Key hyper parameter analysis

Table A.8 presents a hyperparameter study on reconstruction and forecasting tasks for *Navier-Stokes* dataset. Three hyperparameters—initial weight w_0, latent dimension d_z, and width—are varied to observe their impact. We can notice that:

- $w_0 = 30$ slightly improves the reconstruction on the test set.
- $d_z = 64$ yields a better forecasting In-t performance.
- width $= 256$ significantly improves the model's performance across nearly all metrics.

Table A.8: Hyper parameter study - Reconstruction and forecasting results on *Navier-Stokes* dataset. Metrics in MSE. Reconstruction are reported on the Train (*In-t*) and on the Test (*In-t* + *Out-t*).

Param ↓	Value ↓	Reconstruction		Forecasting	
		Train	*Test*	*In-t*	*Out-t*
w_0	20	3,62e-5	6,86e-5	2,78e-4	1,88e-3
	30	3,66e-5	**5,85e-5**	4,03e-4	2,28e-3
d_z	64	3,94e-5	1,11e-4	**1,22e-4**	1,42e-3
	256	2,73e-5	8,03e-5	1,63e-4	2,12e-3
width	64	1,50e-4	2,87e-4	2,84e-4	2,39e-3
	256	**1,60e-5**	6,41e-5	1,23e-4	2,04e-3
CORAL baseline	-	1.05e-4	1.21e-4	1.86e-4	**1.02e-3**

A.4 Supplementary results for geometry-aware inference

A.4.1 Inverse Design for NACA-airfoil

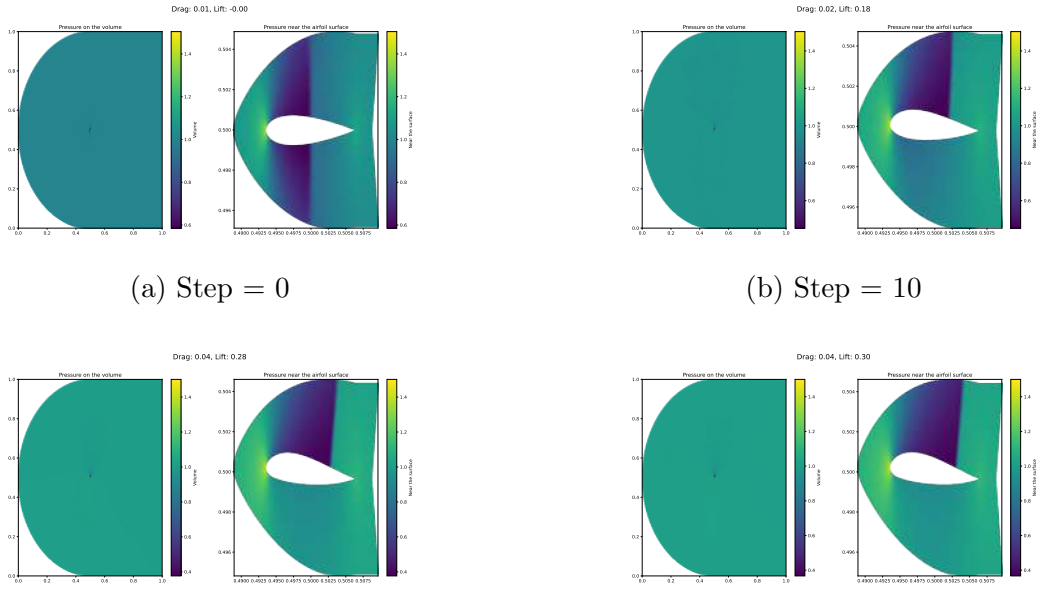
Once trained as a surrogate model to infer the pressure field on *NACA-Euler*, CORAL can be used for the inverse design of a NACA airfoil. We consider an airfoil's shape parameterized by seven spline nodes and wish to minimize drag and maximize lift. We optimize the design parameters in an end-to-end manner. The spline nodes create the input mesh, which CORAL maps to the output pressure field. This pressure field is integrated to compute the drag and the lift, and the loss objective is the squared drag over lift ratio. As can be seen in Figure A.7, iterative optimization results in an asymmetric airfoil shape, enhancing progressively the lift coefficient in line with physical expectations. At the end of the optimization we reach a drag value of 0.04 and lift value of 0.30. Note that CORAL can converge to this solution with only 50 gradient iterations.

A.5 Qualitative results

In this section, we show different visualization of the predictions made by CORAL on the three considered tasks in this paper.

A.5.1 Initial Value Problem

We provide in Figure A.8 and Figure A.9 visualizations of the inferred values of CORAL on *Cylinder* and *Airfoil*.



(a) Step = 0 (b) Step = 10

(c) Step = 30 (d) Step = 50

Figure A.7: Design optimization of a NACA-Airfoil. We use CORAL, which has been trained to infer the pressure from the input mesh, as a surrogate model to compute the drag and lift forces on the airfoil. The whole optimization procedure is done end-to-end, and we optimize the 7 parameters in order to maximize the lift and minimize the drag. We use Adam optimizer.

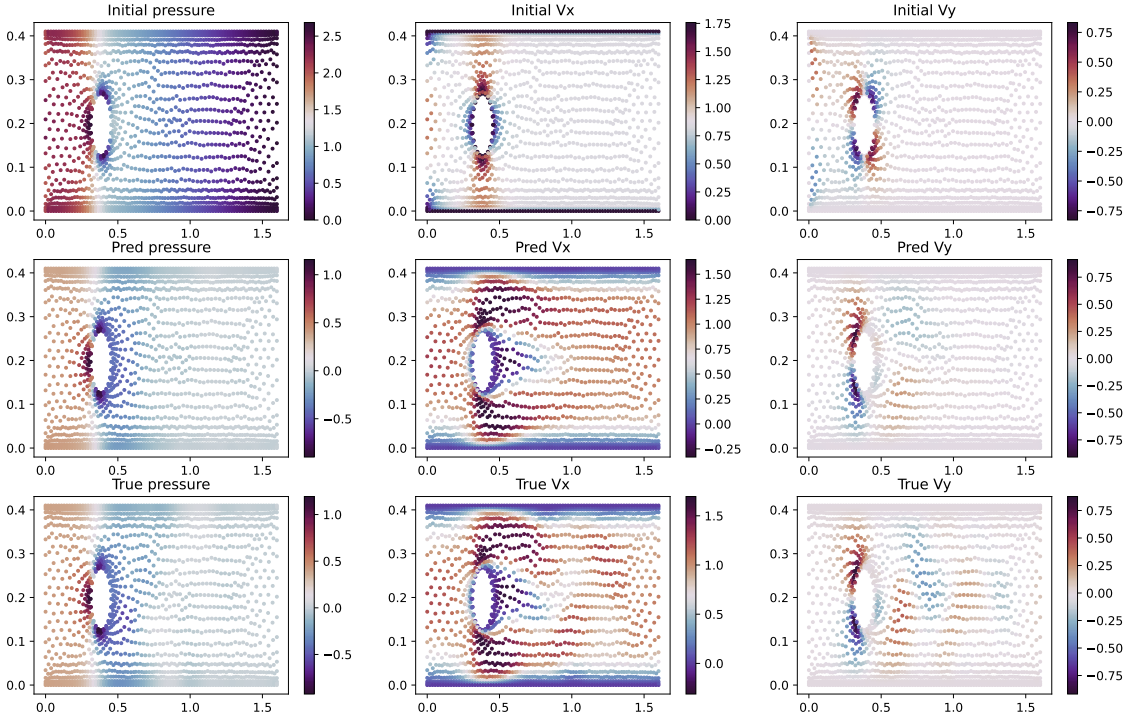


Figure A.8: CORAL prediction on *Cylinder*

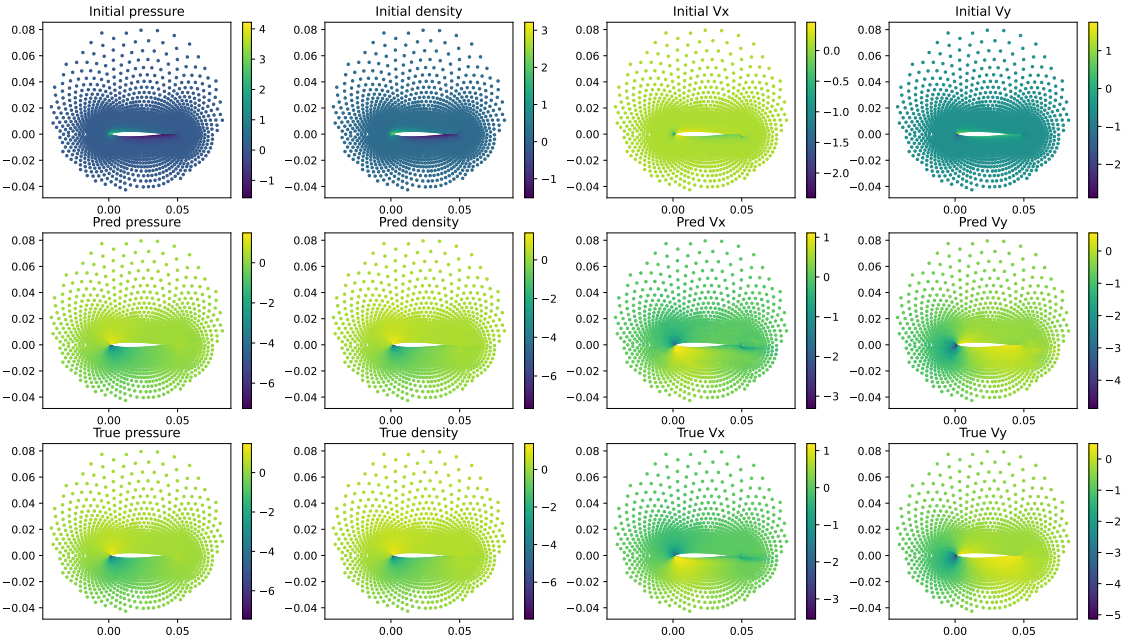


Figure A.9: CORAL prediction on *Airfoil*

A.5.2 Dynamics modeling

We provide in Figure A.11 and Figure A.10 visualization of the predicted trajectories of CORAL on *Navier-Stokes* and *Shallow-Water*.

A.5.3 Geometry-aware inference

We provide in Figure A.12, Figure A.13, Figure A.14 visualization of the predicted values of CORAL on *NACA-Euler*, *Pipe* and *Elasticity*.

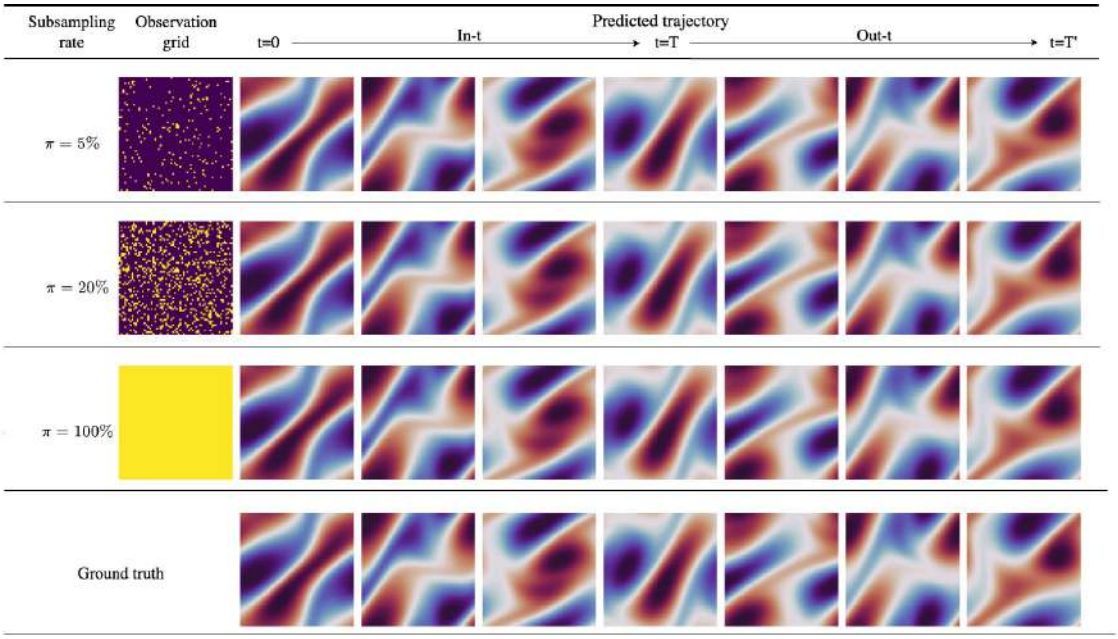


Figure A.10: Prediction MSE per frame for CORAL on *Navier-Stokes* with its corresponding training grid $\mathcal{X}$. Each row corresponds to a different sampling rate and the last row is the ground truth. The predicted trajectory is predicted from $t = 0$ to $t = T'$. In our setting, $T = 19$ and $T' = 39$.

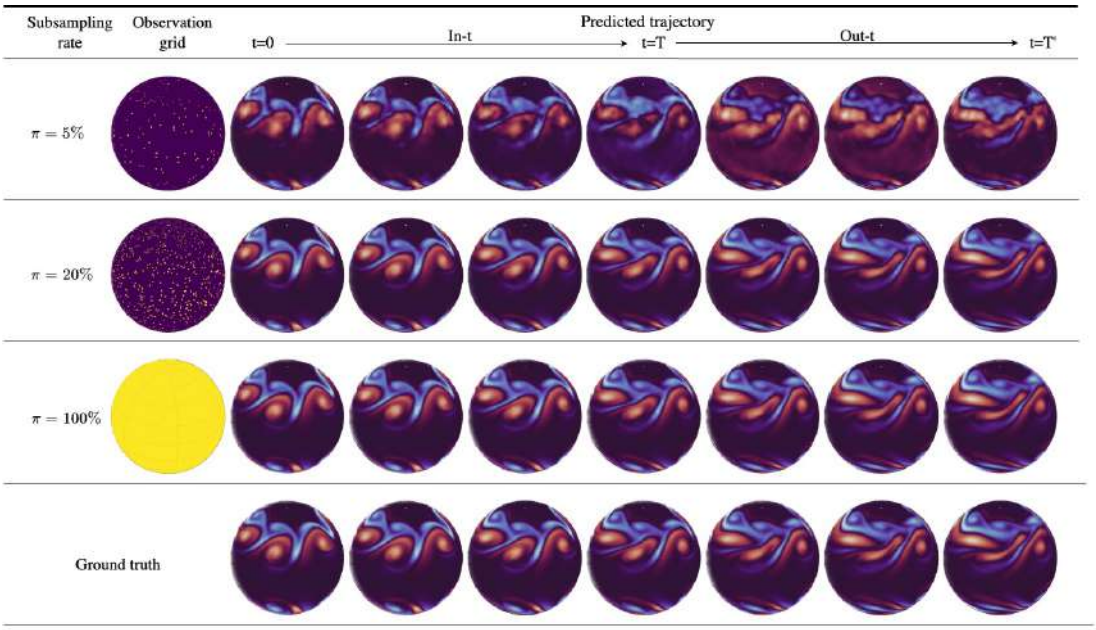


Figure A.11: Prediction MSE per frame for CORAL on *Shallow-Water* with its corresponding training grid $\mathcal{X}$. Each row corresponds to a different sampling rate and the last row is the ground truth. The predicted trajectory is predicted from $t = 0$ to $t = T'$. In our setting, $T = 19$ and $T' = 39$.

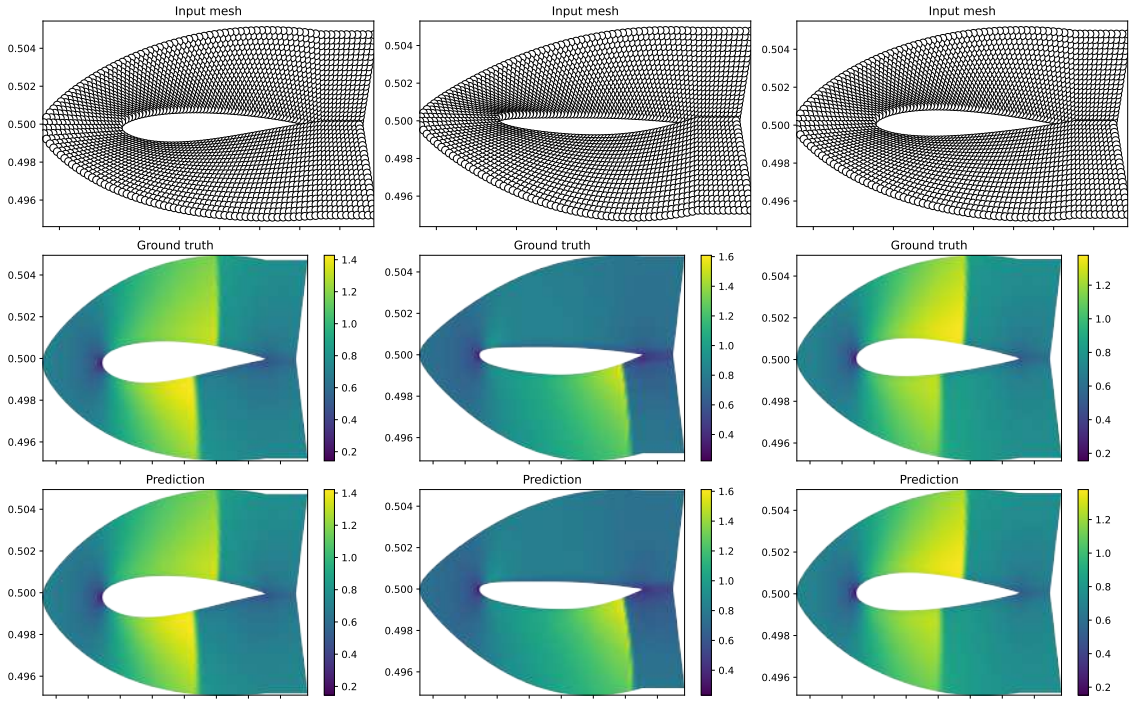


Figure A.12: CORAL prediction on *NACA-Euler*

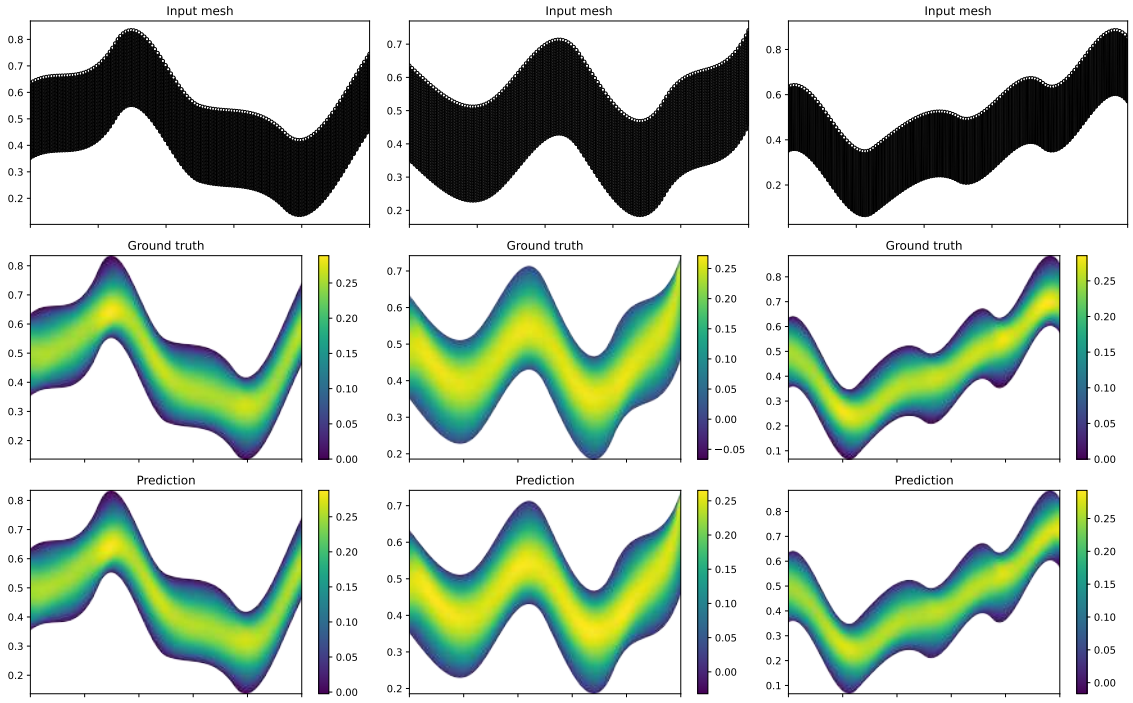


Figure A.13: CORAL prediction on *Pipe*

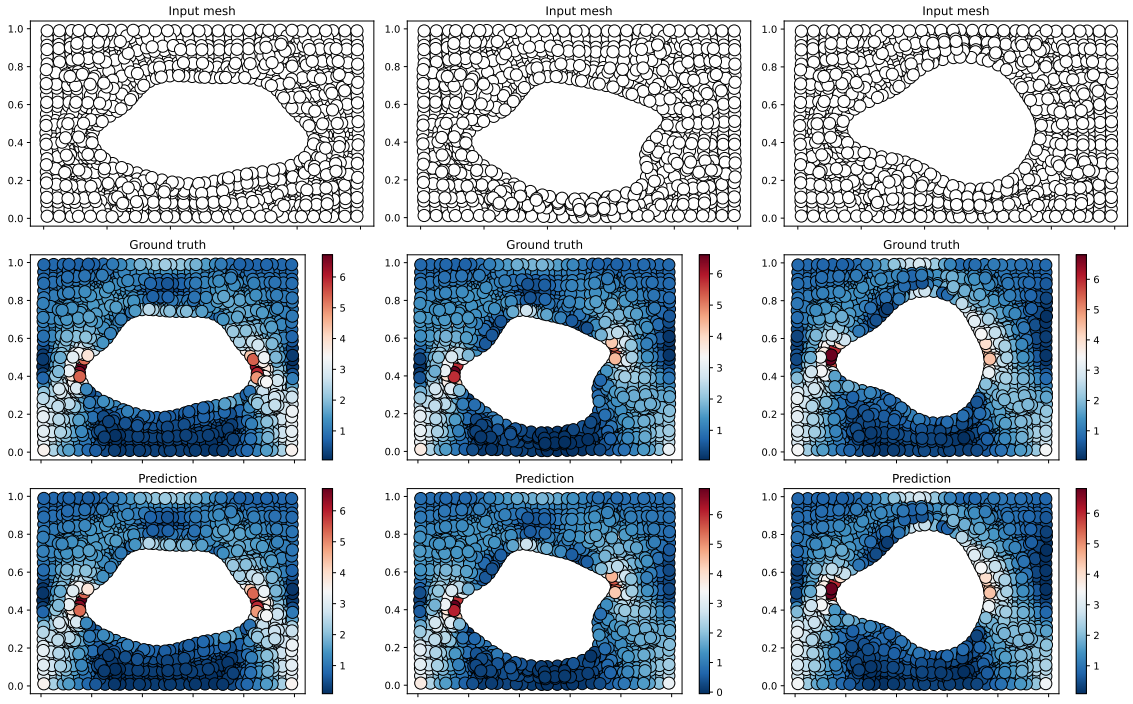


Figure A.14: CORAL prediction on *Elasticity*

Appendix B

Appendix of Chapter 7

B.1 Extended Related Work

Diffusion models for PDE Recently, diffusion models have experienced significant growth and success in generative tasks, such as image or video generation (Ho et al., 2020). This success has motivated their application to physics prediction. Rühling Cachay et al. (2023) propose DYffusion, a framework that adapts the diffusion process to spatio-temporal data for forecasting on long-time rollouts, by performing diffusion-like timesteps in the physical time dimension. PDE-Refiner (Lippe et al., 2023) is a CNN-based method that uses diffusion to stabilize prediction rollouts over long trajectories. Compared to these methods, we perform diffusion in a latent space, reducing the computational cost; and leverage the advanced modeling capabilities of transformers.

Local Neural Fields We are not the first work that proposes to leverage locality to improve the design of neural fields. In a different approach, Bauer et al. (2023) proposed a grid-based latent space where the modulation function ϕ is dependent on the query coordinate x. This concept enables the application of architectures with spatial inductive biases for generation on the latent representations, such as a U-Net Denoiser for diffusion processes. Similarly, Lee et al. (2023) developed a locality-aware, generalizable Implicit Neural Representation (INR) with demonstrated capabilities in generative modeling. Both of these architectures assume regular input structures, be it through patching methods or grid-based layouts.

B.2 Implementation details

Diffusion transformer We illustrate how our diffusion transformer is trained and used at inference in Figure B.1 and Figure B.2. We provide the diffusion step k which acts as a conditioning input for the diffusion model. We use an exponential

decrease for the noise level as in Lippe et al. (2023) i.e. $\alpha_k = 1 - \sigma_{\min}^{k/K}$. We use the same diffusion transformer block as in Peebles and Xie (2023), which relies on amplitude and shift modulations from the diffusion timestamp k:

$$\alpha^{(1)}, \beta^{(1)}, \gamma^{(1)} \leftarrow \text{MLP}_1(k) \tag{B.1}$$

$$\alpha^{(2)}, \beta^{(2)}, \gamma^{(2)} \leftarrow \text{MLP}_2(k) \tag{B.2}$$

$$\boldsymbol{Z}_{[l+1]} \leftarrow \boldsymbol{Z}_{[l]} + \alpha^{(1)} \cdot \texttt{Attention}(\gamma^{(1)} \cdot \texttt{LayerNorm}(\boldsymbol{Z}_{[l]}) + \beta^{(1)}) \tag{B.3}$$

$$\boldsymbol{Z}_{[l+1]} \leftarrow \boldsymbol{Z}_{[l+1]} + \alpha^{(2)} \cdot \texttt{FFN}(\gamma^{(2)} \cdot \texttt{LayerNorm}(\boldsymbol{Z}_{[l+1]} + \beta^{(2)}) \tag{B.4}$$

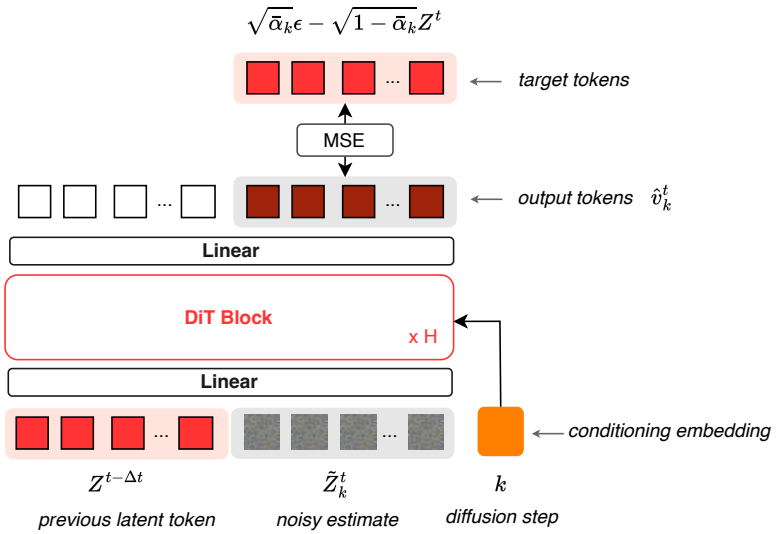


Figure B.1: During training, we noise the next-step latent tokens $\boldsymbol{Z}^{t+\Delta t}$ and train the transformer to predict the "velocity" of the noise. Each DIT block is implemented as in Peebles and Xie (2023).

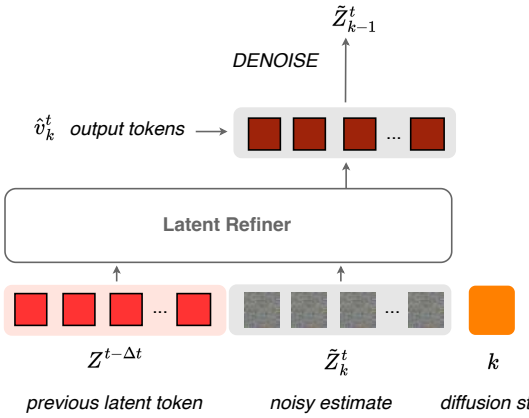


Figure B.2: At inference, we start from $\tilde{\boldsymbol{Z}}_K^{t+\Delta t} \sim \mathcal{N}(0, I)$ and reverse the diffusion process to denoise our prediction. We set our prediction $\hat{\boldsymbol{Z}}^{t+\Delta t} = \tilde{\boldsymbol{Z}}_0^{t+\Delta t}$.

Encoder-Decoder We provide a more detailed description of the encoder-decoder pipeline in Figure B.3.

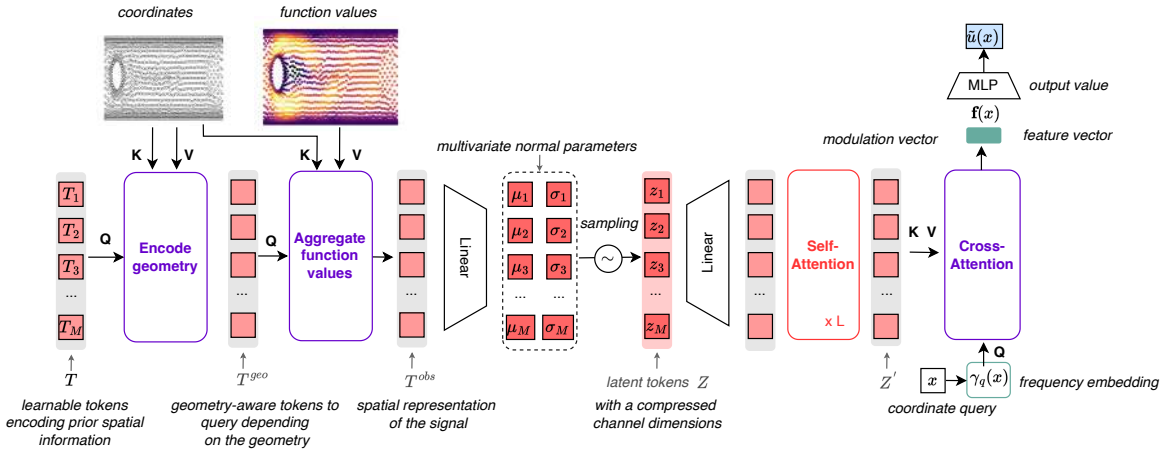


Figure B.3: Architecture of our encoder and decoder. We regularize the architecture as a variational auto-encoder. Cross-attention layers are used to aggregate the N observations into M latent tokens, and to expand the M processed tokens to the queried values. We use a bottleneck layer to reduce the channel dimension of the latent space.

Local INR We show the implementation of our local INR, both with single-band frequency and multi-band frequency, in Figure B.4 and Figure B.5. The cross-attention mechanism enables to retrieve a local feature vector $\mathbf{f}_q(x)$ for each query position x. We then use an MLP to decode this feature vector to retrieve the output value. In practice, we retrieve several feature vectors corresponding each

to separate frequency bandwidths. In this case, we concatenate the feature vectors before decoding them with the MLP.

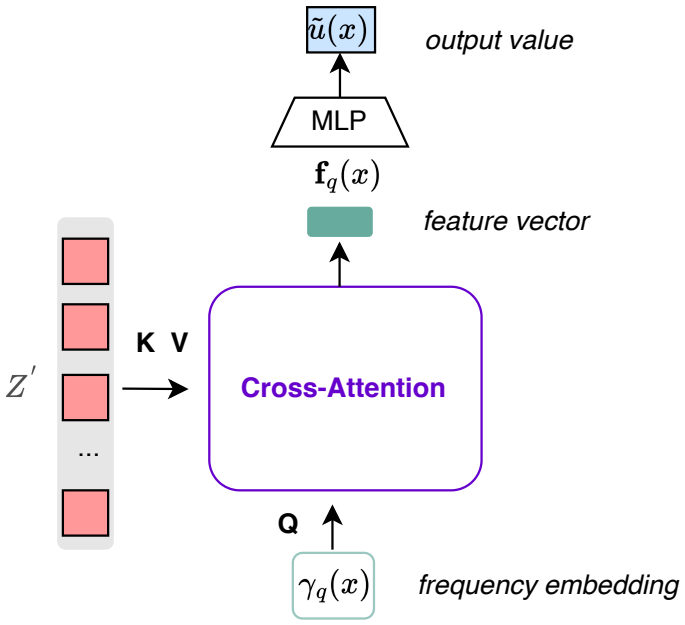


Figure B.4: Single-band local INR decoder of AROMA

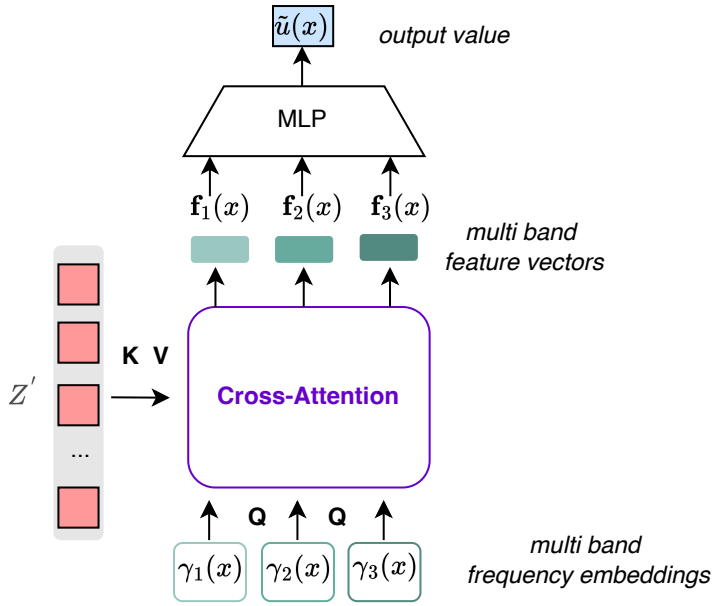


Figure B.5: Multi-band local INR decoder of AROMA

B.2.1 Hyperparameters

We detail the values of the hyperparameters used on each dataset: Table B.2 presents the hyperparameters of the Encoder-Decoder, while Table B.1 presents the hyperparameters of the Diffusion Transformer. We use a cosine scheduler for the tuning learning rate for both trainings, with an initial maximum learning rate of 10^{-3} annealing to 10^{-5} . All experiments were performed with an NVIDIA TITAN RTX.

For the diffusion transformer, we use $K = 3$ diffusion steps for all experiments and only vary the minimum noise σ_{min}.

Table B.1: Diffusion Transformer Hyperparameters for Different Datasets

Hyperparameters	Burgers	NS1e-3	NS1e-4	NS1e-5	Shallow-water	Cylinder-Flow	Airfoil-Flow
hidden_size	128	128	128	128	128	128	128
depth	4	4	4	4	4	4	4
num_heads	4	4	4	4	4	4	4
mlp_ratio	4.0	4.0	4.0	4.0	4.0	4.0	4.0
min_noise	1e-2	1e-2	1e-3	1e-3	1e-3	1e-3	1e-3
denoising_steps	3	3	3	3	3	3	3
epochs	2000	2000	2000	2000	2000	2000	2000

For the encoder-decoder, we have the following hyperparameters:

- hidden_dim: The number h of neurons at each hidden layer.

- num_self_attentions: The number of Self Attention layers used for the decoder.
- num_latents: The number M of latent tokens used to spatially project the objervations and geometries.
- latent_dim: The dimension c of each latent token.
- latent_heads: The number of heads use for the Self Attention layers.
- latent_dim_head: The dimension of each head in a Self Attention layer.
- cross_heads: The number of heads use for the Cross Attention layers.
- cross_dim_head: The dimension of each head in a Cross Attention layer.
- dim: The number of neurons used in the MLP decoder.
- depth_inr: The number of layers in the MLP decoder.
- frequencies: The different frequencies used for the local INR. We use base 2 for all experiments and select 16 frequencies in logarithmic scale per level. For example, [3, 4, 5] means that we construct 3 frequency embedding vectors, the first $\gamma_1(x) = (\cos(2^0\pi x), \sin(2^0\pi x), \ldots, \cos(2^3\pi x), \sin(2^3\pi x))$, for the second $\gamma_2 = (\cos(2^3\pi x), \sin(2^3\pi x), \ldots, \cos(2^4\pi x), \sin(2^4\pi x))$, and for the third $\gamma_3 = (\cos(2^4\pi x), \sin(2^4\pi x), \ldots, \cos(2^5\pi x), \sin(2^5\pi x))$
- dropout_sequence: The ratio of points that are ignored by the encoder.
- feature_dim: The dimension of the feature vector.
- encode_geo: If we use a cross-attention block to encode the geometry.
- max_encoding_freq: The maximum frequency used for the frequency embedding γ of the encoder.
- kl_weight: The weight β used for the VAE training.
- epochs: Number of training epochs.

The most important hyperparameter of the encoder-decoder is the number of tokens M that are used to aggregate the observations and geometries. We show the impact it has on the quality of reconstructions in Table B.3.

Table B.2: Hyperparameters of the Encoder-Decoder for Different Datasets

Hyperparameters	Burgers	NS1e-3	NS1e-4	NS1e-5	Shallow-water	Cylinder-Flow	Airfoil-Flow
hidden_dim	128	128	128	128	128	128	128
num_self_attentions	2	2	2	3	2	2	3
num_latents	32	32	256	256	32	64	64
latent_dim	8	16	16	16	16	16	16
latent_heads	4	4	4	4	4	4	4
latent_dim_head	32	32	32	32	32	32	32
cross_heads	4	4	4	4	4	4	4
cross_dim_head	32	32	32	32	32	32	32
dim	128	128	128	128	64	128	128
depth_inr	3	3	3	3	3	3	3
frequencies	[3, 4, 5]	[2, 3]	[3, 4, 5]	[3, 4, 5]	[2, 3]	[3, 4, 5]	[3, 4, 5]
dropout_sequence	0.1	0.1	0.1	0.1	0.1	0.1	0.1
feature_dim	16	16	16	16	16	16	16
encode_geo	False	True	False	False	True	True	True
max_encoding_freq	4	4	4	4	5	4	5
kl_weight	1e-4	1e-4	1e-4	1e-5	1e-5	1e-5	1e-5
epochs	5000	5000	5000	5000	5000	5000	5000

B.3 Additional results

B.3.1 Time complexity analysis

We denote N as the number of observations of $\boldsymbol{u}$, M as the number of tokens used to compress the information, T as the number of autoregressive calls in the rollout, K as the number of refinement steps, and d as the number of channels used in the attention mechanism. The most computationally expensive operations in our architecture are the cross-attention and self-attention blocks. For simplification, we omit the geometry encoding block in this study.

The cost of the cross-attention in the encoder is $O(NMd)$, and similarly, the cost of the cross-attention in the decoder is $O(NMd)$. Let L_1 and L_2 represent the number of layers in the decoder and diffusion transformer, respectively. The cost of the self-attention layers in the decoder is $O(L_1M^2d)$, while in the diffusion transformer, it is $O(4L_2M^2d)$.

To unroll the dynamics, we encode the initial condition, obtain the predictions in the latent space, and then decode in parallel, yielding a total cost of $O((2N + 4KTL_2M + L_1M)Md)$. As expected, our architecture has linear complexity in the number of observations through the cross-attention layers. In contrast, GNOT relies on linear attention, resulting in a time complexity of $O((LN)d^2)$ for each prediction, where L is the depth of the network. At inference, the cost per step along a trajectory is LNd^2 for GNOT, compared to $4KL_2M^2d$ for AROMA.

For instance, using $K = 3$, $M = 64$, $N = 4096$, and $d = 128$, GNOT's cost is approximately 10 times that of AROMA for each prediction throughout the rollout. Therefore AROMA is more efficient when $M \ll N$.

B.3.2 Encoding interpretation

We provide in Figure B.6 a qualitative analysis through cross-attention visualizations how the geometry encoding block helps to capture the geometry of the domain. In the first cross-attention block, the query tokens $\mathbf{T}$ are not aware of the geometry and therefore attend to large regions of the domains. This lets the model understand, where the boundaries of the domain are and therefore where the cylinder is. Once the query tokens have aggregated the mesh information, the cross attention between $\mathbf{T}^{\text{geo}}$ and the positions are sharper and depend on the geometry.

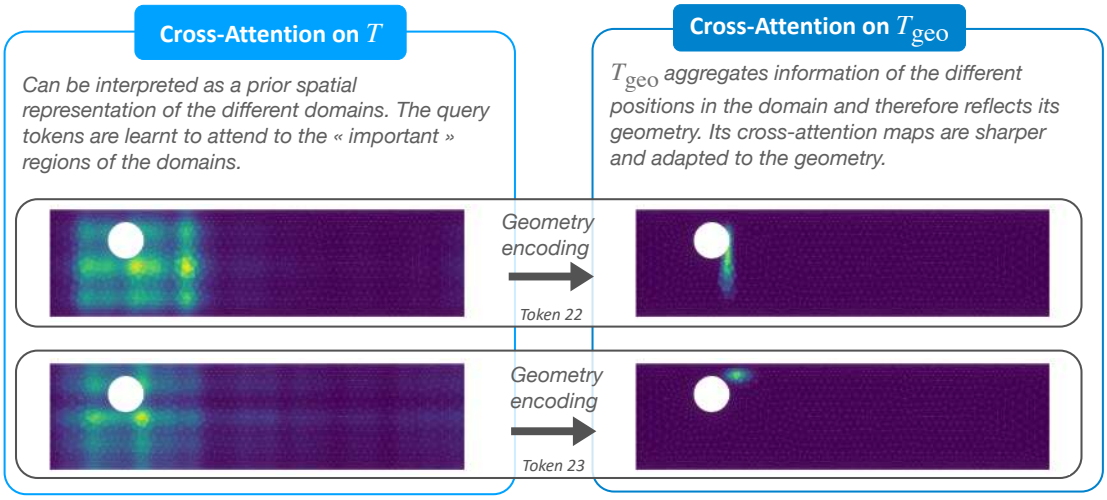


Figure B.6: Evolution of the cross-attention maps between the geometry encoding stage and the observation encoding stage. Blue means the cross-attention value is close to zero while yellow means the cross-attention score is close to one.

B.3.3 Example rollouts

We show examples of rollout predictions using AROMA on *Burgers* dataset in Figure B.7, on *Navier-Stokes* 1×10^{-3} dataset in Figure B.9 and on *CylinderFlow* in Figure B.10. AROMA returns predictions that remain stable and accurate, even outside the training time horizon.

For *Navier-Stokes*, we show an example of test trajectory in the training horizon (Figure B.9a) and in extrapolation (Figure B.9b).

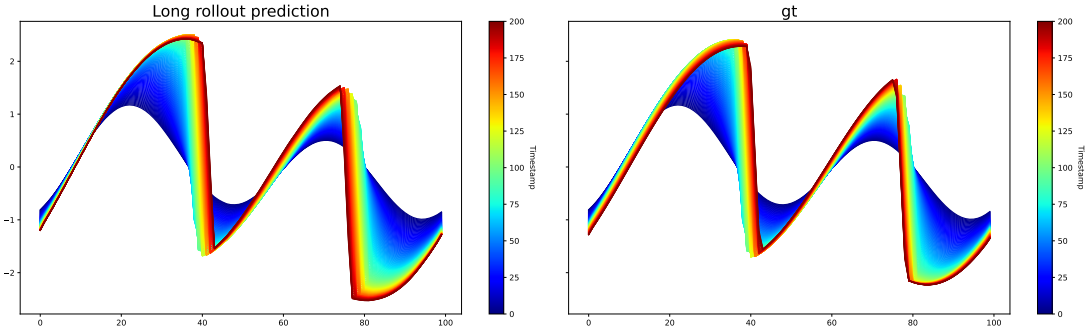


Figure B.7: Test example of a long rollout trajectory with AROMA on *Burgers*. Left is the predicted trajectory and right is the ground truth.

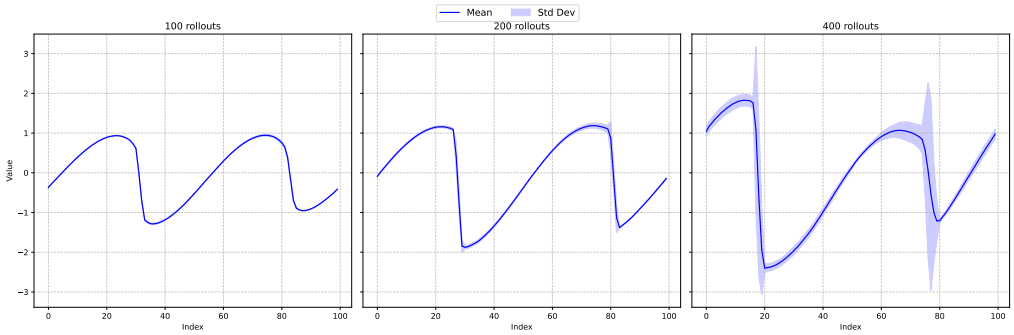
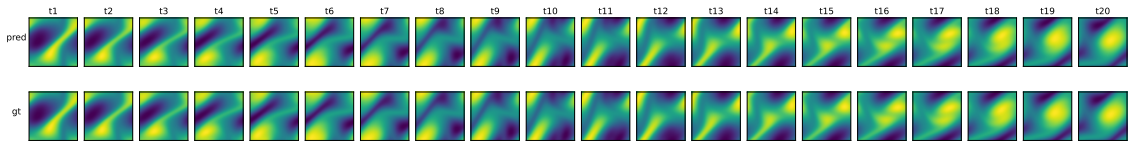
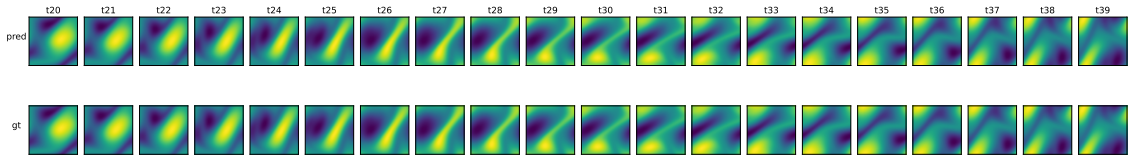


Figure B.8: Uncertainty of AROMA over rollout steps. The blue line is the mean prediction while the blue shade represents the mean $\pm$ 3 $\times$ standard deviation.



(a) *In-t*



(b) *Out-t*

Figure B.9: Test example rollout trajectories with AROMA on *Navier-Stokes* 1×10^{-3}. **Top:** predicted trajectory on *In-t*. **Bottom:** trajectory on *Out-t*. First row in each subfigure shows the prediction, the second row shows the ground truth.

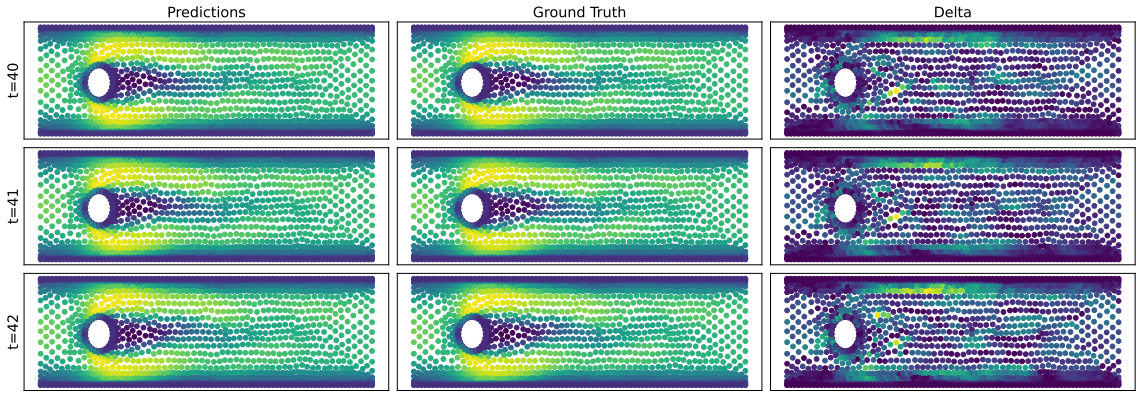
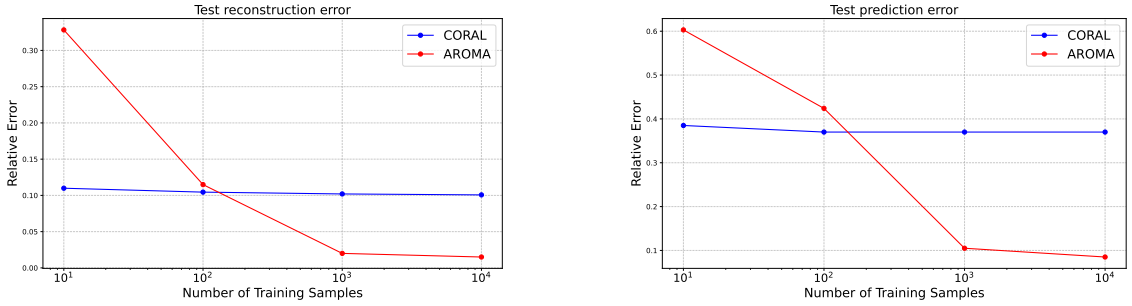


Figure B.10: Visualization of AROMA's predictions on *Cylinder* for (*Out-t*). The left panel shows the prediction, the middle panel displays the ground truth, and the right panel is the point-wise error.

B.3.4 Scaling experiments

In Figure B.11, we compare the reconstruction and prediction capabilities of CORAL and AROMA on *Navier-Stokes* 1×10^{-4} given the number of training trajectories. As evidenced, our architecture outperforms CORAL significantly when the number of trajectories is greater than 10^3, highlighting its efficacy in handling large amounts of data..



(a) Step 1: Autoencoding

(b) Step 2: Rollout prediction

Figure B.11: Scaling comparison of AROMA & CORAL: relative L_2 error with respect to the number of training trajectories

B.3.5 Spatial tokens perturbation analysis

To validate the spatial interpretation of our latent tokens, we establish a baseline code $\boldsymbol{Z}^0$, and introduce perturbations by sequentially replacing the j-th token, $\boldsymbol{z}_j^0$, with subsequent tokens along the trajectory, denoted as $\boldsymbol{z}_j^1, \boldsymbol{z}_j^2, \ldots, \boldsymbol{z}_j^t$. Thus, the perturbed tokens mirror $\boldsymbol{Z}^0$ in all aspects except for the j-th token, which evolves according to the true token dynamics. We show reconstruction visualizations of

the perturbed tokens in Figures B.12 to B.19. On the right side, we show the groundtruth of the trajectory. On the left side, is the change in AROMA's prediction in response to the token perturbation. These figures show that the perturbation of a token only impacts the reconstructed field locally, which validates the spatial structure of our tokens. Additionally, we can notice some interesting effects of the token perturbations near the boundaries in Figures B.14 and B.19: our encoder-decoder has discovered from data and without explicit supervision that the solutions had periodic boundary conditions by leveraging the encoded geometry and the function values. This validates the architecture of our cross-attention module between the function values, the spatial coordinates and the geometry-aware tokens.

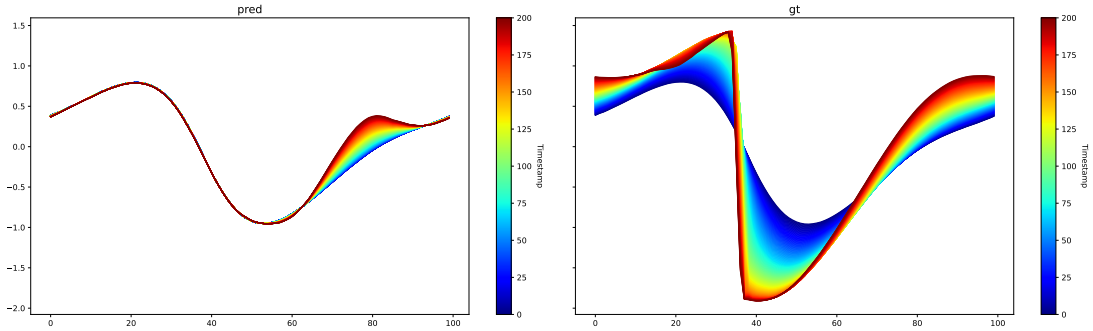


Figure B.12: Perturbation analysis on *Burgers*. Token 0.

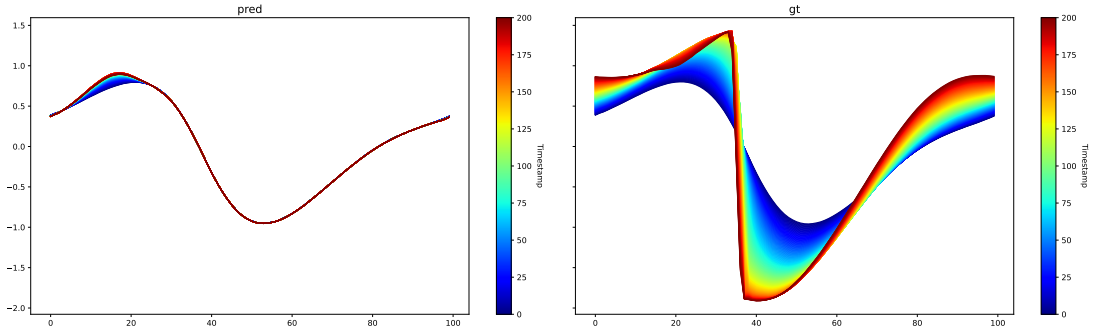


Figure B.13: Perturbation analysis on *Burgers*. Token 1.

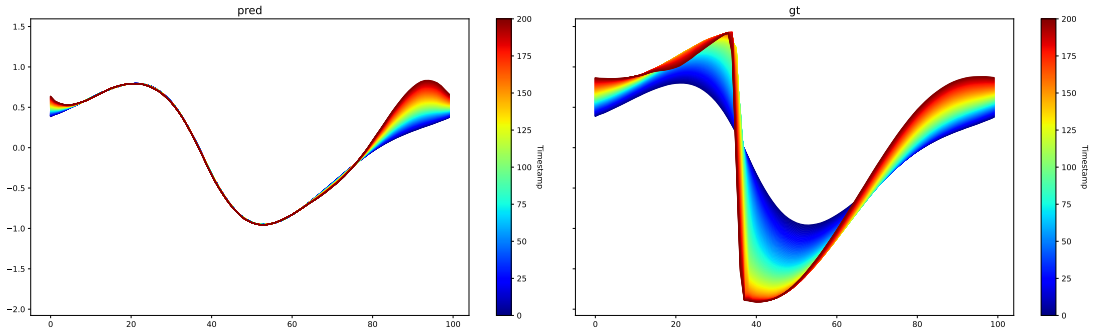


Figure B.14: Perturbation analysis on *Burgers*. Token 2.

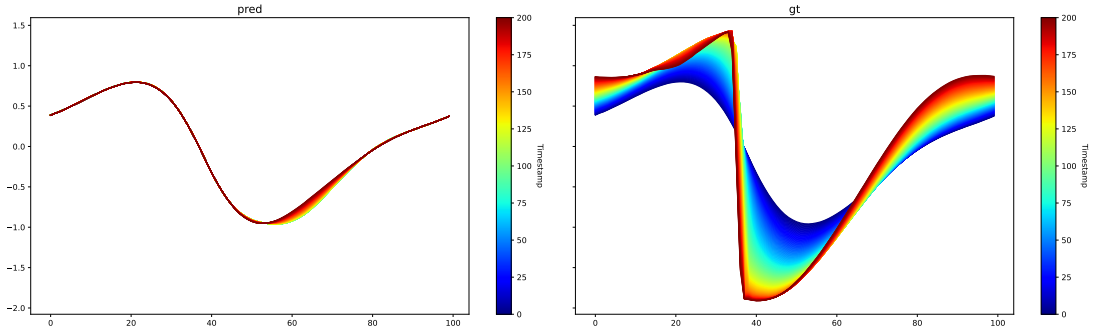


Figure B.15: Perturbation analysis on *Burgers*. Token 3.

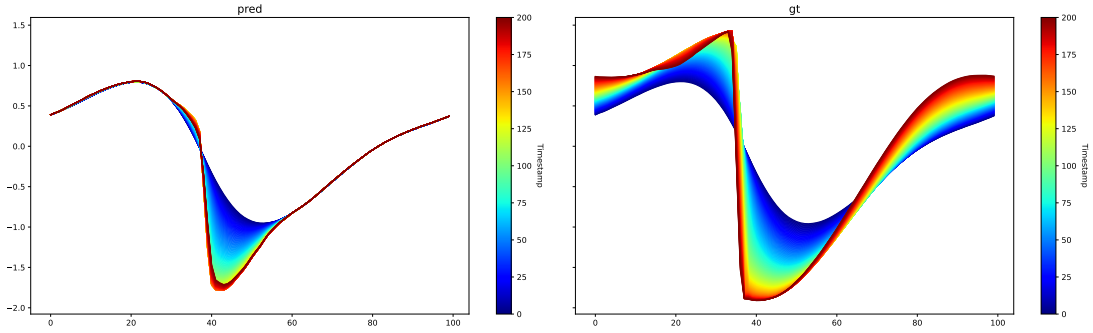


Figure B.16: Perturbation analysis on *Burgers*. Token 5.

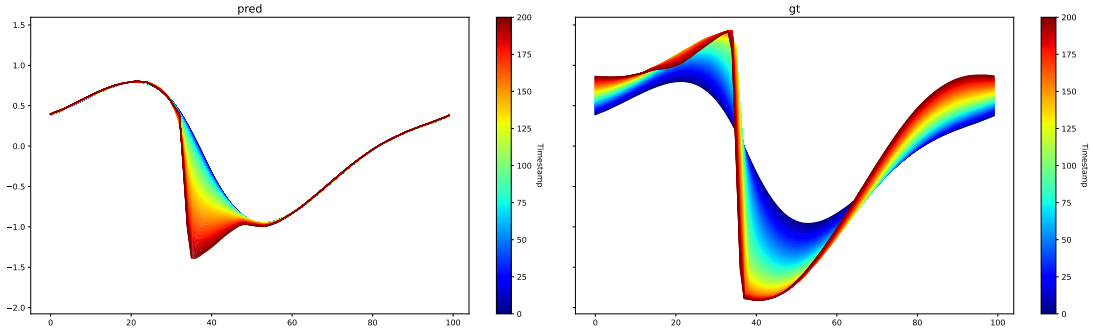


Figure B.17: Perturbation analysis on *Burgers*. Token 6.

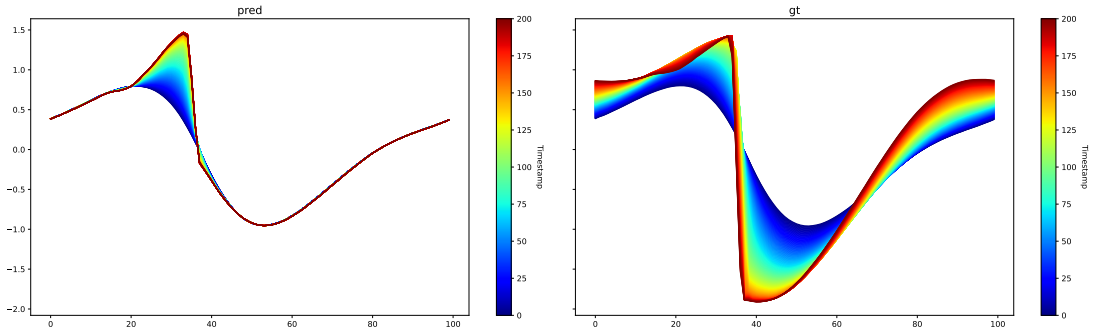


Figure B.18: Perturbation analysis on *Burgers*. Token 7.

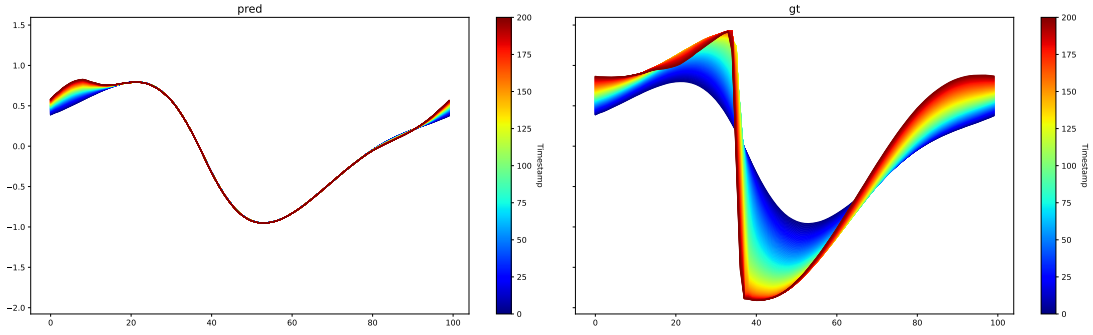


Figure B.19: Perturbation analysis on *Burgers*. Token 8.

B.3.6 Ablation studies

Number of tokens M We show the impact of the number of latent tokens on the *Navier-Stokes* 1×10^{-4} dataset in Table B.3. We train our auto-encoder with 10000 trajectories. We can see that the performance increases with the number of tokens.

#Latent Tokens	Test Reconstruction error
64	0.02664
128	0.0123
256	0.01049

Table B.3: Influence of the number of latent tokens on the test reconstruction capabilities on *Navier-Stokes* 1×10^{-4}. Performance in Relative L_2 Error.

Auto-encoding vs VAE Our framework can also be used without the KL regularization, and could potentially be employed with other forms of regularization, such as L2 regularizaton or vector-quantization (Oord et al., 2017). We investigated in Table B.4 the impact the KL regularization had on the overall rollout performance, and selected an autoencoder with L2 regularization (weight decay) as baseline. Our conclusion is that using an autoencoder with L2 regularization is a viable alternative to the VAE in some cases for achieving a smooth latent space. The autoencoder demonstrated superior performance on two datasets (*Burgers* and *Navier-Stokes* 1×10^{-4}, explained by its lower reconstruction errors, which translate into better rollout performance. However, for the more challenging *Navier-Stokes* 1×10^{-5} case, the autoencoder's latent space exhibited high variance, which may explain the observed performance difference with the VAE.

No-diffusion vs diffusion As an ablation, we also measured the influence of the diffusion formulation on the rollout accuracy by comparing to the same transformer architecture trained directly with an MSE on the mean tokens. The deterministic version of AROMA shows consistently robust performance and even surpasses the diffusion version on the *Navier-Stokes* 1×10^{-4} case (Table B.4). This demonstrates that the latent tokens obtained with AROMA contain meaningful information for dynamics modeling. On the other hand, the deterministic version yields less accurate long rollouts on *Burgers* or *KS* in Figure B.20 and Figure 7.3. Note that using diffusion allows us to model the trajectory distribution, which opens the way to infer statistics on this distribution. This is key, for example, when modeling uncertainty, which is a critical problem for these models.

Latent MLP vs Latent Transformer Modeling interactions at the local and global levels is key to learn the dynamics faithfully. Experiments using MLPs (Table B.4) as time steppers which do not consider interactions between tokens lead to significantly lower performance compared to transformers.

Table B.4: Ablation Study. Metrics in Relative L_2 on the test set.

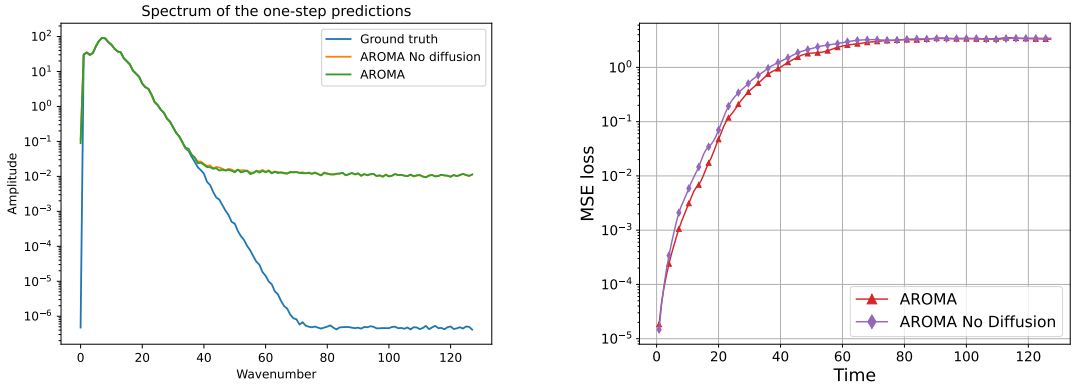
Model	*Burgers*	*Navier-Stokes* 1×10^{-4}	*Navier-Stokes* 1×10^{-5}
AROMA + auto-encoding	$\mathbf{3.43 \times 10^{-2}}$	$\mathbf{5.02 \times 10^{-2}}$	2.10×10^{-1}
AROMA w/o diffusion	4.31×10^{-2}	7.50×10^{-2}	1.28×10^{-1}
AROMA + mlp	1.11×10^{-1}	1.00×10^{0}	8.25×10^{-1}
AROMA	3.65×10^{-2}	1.05×10^{-1}	$\mathbf{1.24 \times 10^{-1}}$

B.3.7 Kuramoto-Sivashinsky : a failure case

We conducted additional experiments on a chaotic 1D PDE, the Kuramoto-Sivashinsky (*KS*) equation. We found that AROMA currently struggles with dynamics that exhibit chaotic phenomena and non-decaying spectra, as shown in Figure B.20. The primary limitation appears to be the reconstruction capabilities of the encoder-decoder. For the *KS* equation, we found that obtaining reconstructions with an MSE in the range of 1e-10 to 1e-12 was necessary for accurate spectrum reconstruction. Like all models leveraging a reduced latent representation space, AROMA inherently loses some of the fine-grained details necessary for accurately capturing chaotic behavior. The diffusion framework slightly improves the high correlation time compared to the deterministic version, however the main bottleneck comes from the decoder (Table B.5). In conclusion, while AROMA performs very well on simpler dynamics, dealing with chaotic phenomena requires more involved modeling that explicitly targets the chaotic component. Note that using dedicated modules for this purpose is a current practice in fluid dynamics - e.g. LES (Large eddy simulation).

Table B.5: Test results on the KS equation. The evaluated metrics include: 1-step prediction MSE, MSE over the entire rollout (160 timestamps), and the duration for which the correlation between the generated samples and the ground truth remains above 0.8.

Baseline	1 step. MSE	Rollout MSE	Corr. ≥ 0.8
AROMA w/o diffusion	$\mathbf{1.25 \times 10^{-5}}$	2.20×10^{0}	32.8s
AROMA	1.81×10^{-5}	$\mathbf{2.07 \times 10^{0}}$	**36.0s**



(a) Spectrum of AROMA's 1-step predictions vs ground truth.

(b) Comparison of the MSE loss (↓) with and without diffusion.

Figure B.20: Qualitative results on KS equation.

B.3.8 Latent space dynamics

For *Navier-Stokes*, we show how the mean (Figure B.21) and standard deviation tokens (Figure B.22) evolve over time for a given test trajectory. We show the predicted trajectory of the latent tokens Z in the latent space in Figure B.23. In practice, the tokens where the logvar is 0 (i.e. a high variance) on Figure B.22 do not impact the prediction (Rolinek et al., 2019). We can therefore see, that ouf of the 16 tokens used, the most influential ones are Token 6, 7, 8, 9, 15, 16, as they clearly exhibit non-noisy patterns.

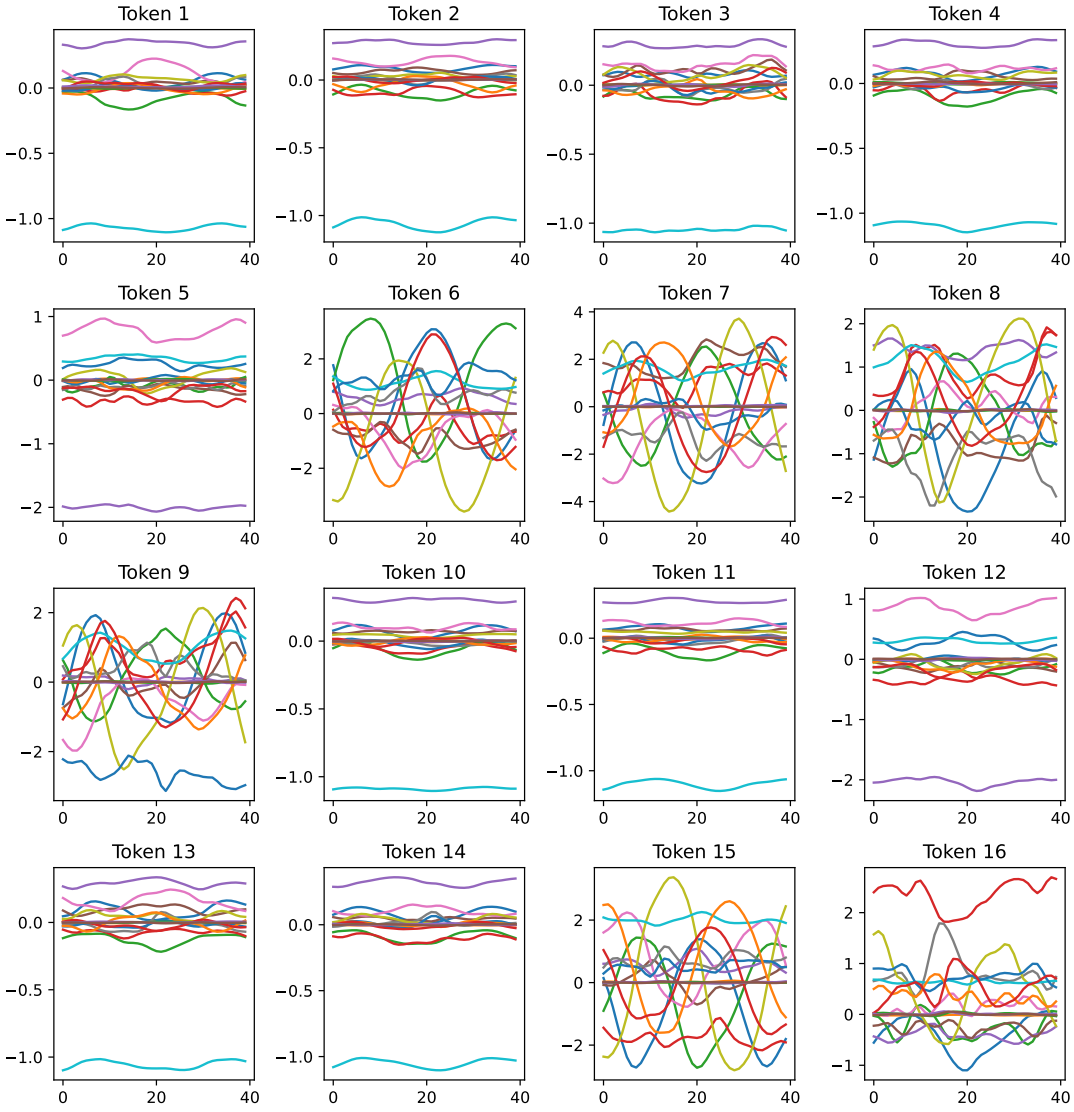


Figure B.21: Latent space dynamics on *Navier-Stokes 1e-3* - Mean tokens over time. Each color line is a different token channel.

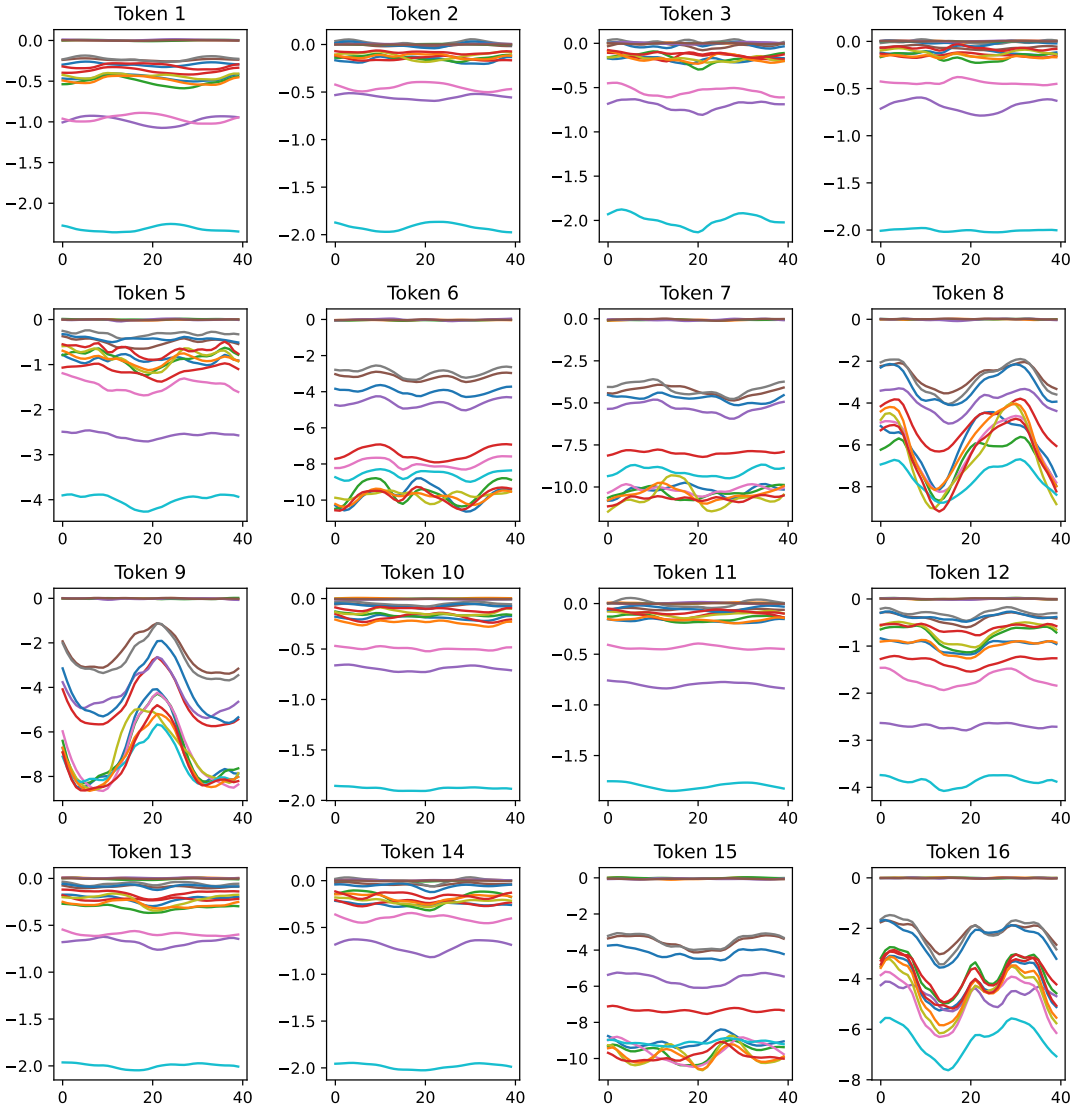


Figure B.22: Latent space dynamics on *Navier-Stokes* - Logvar tokens over time. Each color line is a different token channel.

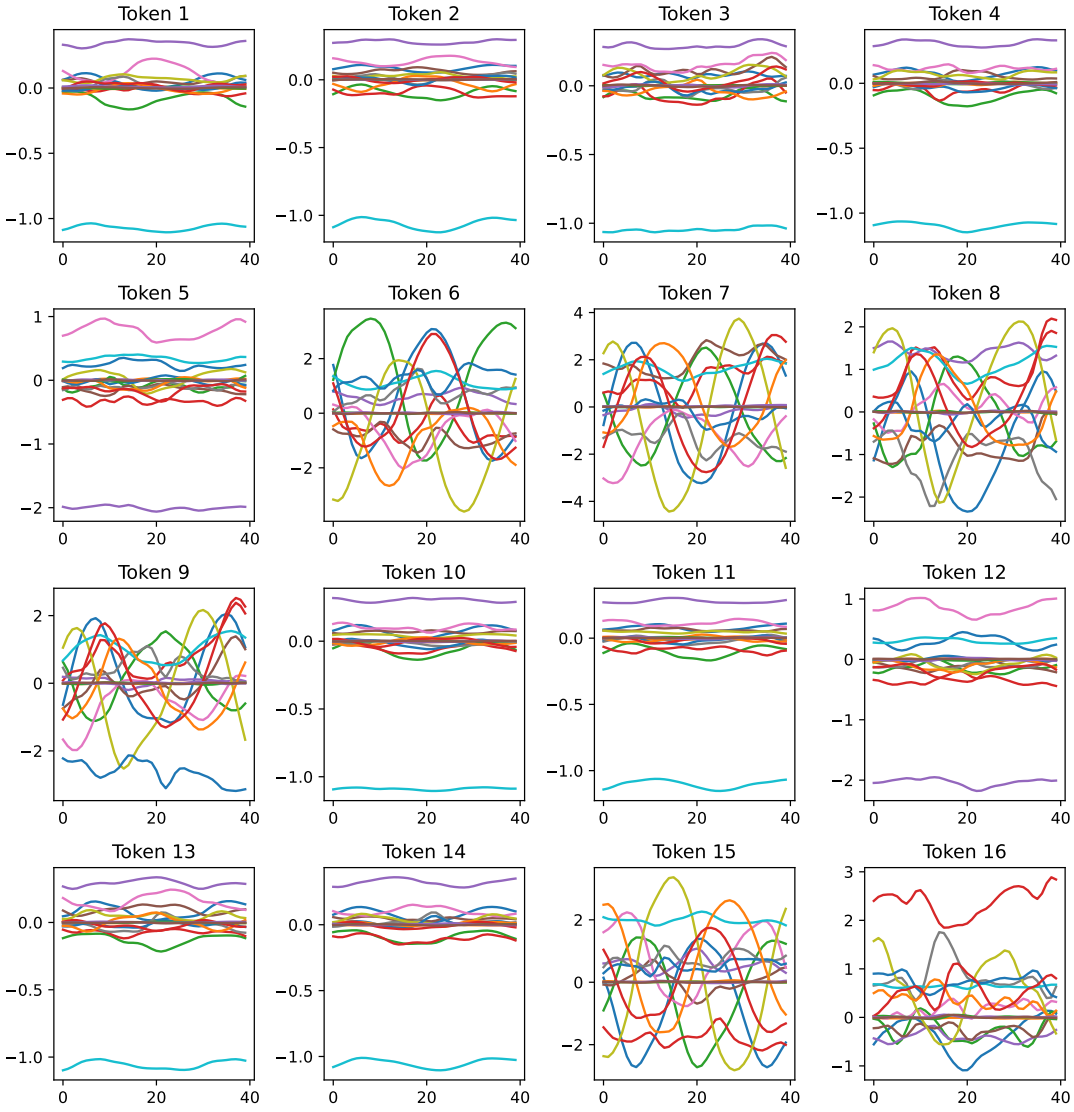


Figure B.23: Latent space dynamics on *Navier-Stokes* - Predicted tokens over time. Each color line is a different token channel.

Appendix C

Appendix of Chapter 8

C.1 Related Work

C.1.1 Learning parametric PDEs

The classical ML paradigm The classical ML paradigm for solving parametric PDEs consists in sampling from the PDE parameter distribution trajectories to generalize to new PDE parameter values. It is the classical ERM approach. The natural way for generalizing to new PDE parameters is to explicitly embed them in the neural network (Brandstetter et al., 2022d). Takamoto et al. (2023) proposed a channel-attention mechanism to guide neural solvers with the physical coefficients given as input; it requires complete knowledge of the physical system and are not designed for other PDE parameter values, e.g., boundary conditions. It is commonly assumed that prior knowledge are not available, but instead rely on past states of trajectories for inferring the dynamics. Neural solvers and operators learn parametric PDEs by stacking the past states as channel information as done in Li et al. (2021), or by creating additional temporal dimension as done in video prediction contexts (Ho et al., 2022; McCabe et al., 2023). Their performance drops when shifts occur in the data distribution, which is often met with parametric PDEs, as small changes in the PDE parameters can lead to various dynamics. To better generalize to new PDE parameter values, Subramanian et al. (2023) instead leverages fine-tuning from pretrained models to generalize to new PDE parameters. It however often necessitates a relatively large number of fine tuning samples to effectively adapt to new PDE parameter values, by updating all or a subset of parameters (Herde et al., 2024; Hao et al., 2024).

Gradient-based adaptation To better adapt to new PDE parameters values at inference, several works have explored learning on multiple environments. During training, a limited number of environments are available, each corresponding to a specific PDE instance. Yin et al. (2022a) introduced LEADS, a multi-task frame-

work for learning parametric PDEs, where a shared model from all environments and a model specific to each environment are learned jointly. At inference, for a new PDE instance, the shared model remain frozen and only a model specific to that environment is learned. Kirchmeyer et al. (2022) proposed to perform adaptive conditioning in the parameter space; the framework adapts the weights of a model to each environment via a hyper-network conditioned by a context vector c^e specific to each environment. At inference, the model adapts to a new environment by only tuning c^e. Park et al. (2023) bridged the gap from the classical gradient-based meta-learning approaches by addressing the limitations of second-order optimization of MAML and its variants (Finn et al., 2017; Zintgraf et al., 2019b). Other works have also extended these frameworks to quantify uncertainty of the predictions : Jiaqi et al. (2024) proposed a conditional neural process to capture uncertainty in the context of multiple environments with sparse trajectories, while Nzoyem et al. (2024) leveraged information from multiple environments to enable more robust predictions and uncertainty quantification.

In-context learning for PDE Inspired by the in-context learning (ICL) paradigm in large language models (LLMs), recent works have explored adapting this approach for solving PDEs and modeling dynamical systems. One of the earliest efforts in this direction is Yang et al. (2023), which aims to learn operators capable of adapting to different physical scenarios by leveraging in-context examples. Their approach utilizes an encoder-decoder transformer, where the transformer encodes the context prompt. This prompt, together with a query, is then passed to the decoder, which predicts the corresponding output values of the state vector. However, since functions are represented as scattered point tokens, the model encounters computational complexity challenges and is primarily limited to 1D ODEs or sparse 2D data. Cao et al. (2024) extends this framework by leveraging vision transformers (ViTs) that operate on image patches, similar to the ViT baseline used in our experiments. However, their evaluation is conducted on a physics setting with limited diversity. Chen et al. (2024) takes a different approach, focusing on unsupervised pretraining for operator learning. Their method involves pretraining the encoder-decoder of neural operators on proxy tasks (such as masked prediction or super-resolution) that require only snapshots of the dynamics rather than full simulation data, followed by fine-tuning on target dynamics. In this case, ICL is used only at inference time, where context examples similar to a query input are retrieved from the training set, and their solutions are aggregated and averaged to form the final prediction. This setting differs significantly from ours. Notably, all these approaches rely on deterministic ViT-like architectures, whereas our method employs a generative stochastic model.

Note that in-context learning is still a not well understood phenomenon and that different hypotheses are being explored which attempt to fill this gap Dong et al.

(2024). Two prevalent explanations come from a Bayesian perspective on ICL as introduced in a popular paper Xie et al. (2022) and the gradient descent view as introduced e.g. in Dai et al. (2023) that identied a dual form between transformer attention and gradient descent highlighting relations between GPT-based ICL and expicit fine tuning.

C.1.2 Generative models

Auto-regressive Transformers for Images and Videos Recent works have explored combining language modeling techniques with image and video generation, typically using a VQ-VAE (Oord et al., 2017) paired with a causal transformer (Esser et al., 2021) or a bidirectional transformer (Chang et al., 2022). VQGAN (Esser et al., 2021) has become the leading framework by incorporating perceptual and adversarial losses to improve the visual realism of decoder outputs from quantized latent representations. However, while these methods succeed in generating visually plausible images, they introduce a bias—driven by perceptual and adversarial losses—that leads the network to prioritize perceptual similarity and realism, often causing reconstructions to deviate from the true input. In contrast, `Zebra` focuses on maximizing reconstruction accuracy, and we did not observe benefits from using adversarial or perceptual losses during training.

In video generation, models like Magvit (Yu et al., 2023a) and Magvit2 (Yu et al., 2023b) adopt similar strategies, using 3D CNN encoders to compress sequences of video frames into spatiotemporal latent representations by exploiting the structural similarities between successive frames in a video. However, such temporal compression is unsuitable for modeling partial differential equations (PDEs), where temporal dynamics can vary significantly between frames depending on the temporal resolution. With `Zebra`, we spatially compress observations using the encoder and learn the temporal dynamics with an auto-regressive transformer, avoiding temporal compression.

C.2 Dataset details

C.2.1 Advection

We consider a 1D advection equation with advection speed parameter β:

$$\partial_t u + \beta \partial_x u = 0$$

For each environment, we sample β with a uniform distribution in $[0, 4]$. We sample 1200 parameters, and 10 trajectories per parameter, constituting a training set of 12000 trajectories. At test time, we draw 12 new parameters and evaluate the performance on 10 trajectories each.

Table C.1: Dataset Summary

Dataset Name	Number of env.	Trajectories per env.	Main parameters
Advection	1200	10	Advection speed
Heat	1200	10	Diffusion and forcing
Burgers	1200	10	Diffusion and forcing
Wave boundary	4	3000	Boundary conditions
Combined equation	1200	10	α, β, γ
Wave 2D	1200	10	Wave celerity and damping
Vorticity 2D	1200	10	Diffusion

We fix the size of the domain $L = 128$ and draw initial conditions as described in Equation (C.2) in Appendix C.2.5 and generate solutions with the method of lines and the pseudo-spectral solver described in Brandstetter et al. (2022d). We take 140 snapshots along a 100s long simulations, which we downsample to 14 timestamps for training. We used a spatial resolution of 256.

C.2.2 Burgers

We consider the Burgers equation as a special case of the combined equation described in Appendix C.2.5 and initially in Brandstetter et al. (2022d), with fixed $\gamma = 0$ and $\alpha = 0.5$. However, in this setting, we include a forcing term $\delta(t, x) = \sum_{j=1}^{J} A_j \sin(\omega_j t + 2\pi \ell_j x/L + \phi_j)$ that can vary across different environments. We fix $J = 5$, $L = 16$. We draw initial conditions as described in Equation (C.2).

For each environment, we sample β with a log-uniform distribution in $[1e-3, 5]$, and sample the forcing term coefficients uniformly: $A_j \in [-0.5, 0.5]$, $\omega_j \in [-0.4, -0.4]$, $\ell_j \in \{1, 2, 3\}$, $\phi_j \in [0, 2\pi]$. We create a dataset of 1200 environments with 10 trajectories for training, and 12 environments with 10 trajectories for testing.

We use the solver from Brandstetter et al. (2022d), and take 250 snapshots along the 4s of the generations. We employ a spatial resolution of 256 and downsample the temporal resolution to 25 frames.

C.2.3 Heat

We consider the heat equation as a special case of the combined equation described in Appendix C.2.5 and initially in Brandstetter et al. (2022d), with fixed $\gamma = 0$ and $\alpha = 0$. However, in this setting, we include a forcing term $\delta(t, x) = \sum_{j=1}^{J} A_j \sin(\omega_j t + 2\pi \ell_j x/L + \phi_j)$ that can vary across different environments. We fix $J = 5$, $L = 16$. We draw initial conditions as described in Equation (C.2).

For each environment, we sample β with a log-uniform distribution in $[1e-3, 5]$, and sample the forcing term coefficients uniformly: $A_j \in [-0.5, 0.5]$, $\omega_j \in$

$[-0.4, -0.4]$, $\ell_j \in \{1, 2, 3\}$, $\phi_j \in [0, 2\pi]$. We create a dataset of 1200 environments with 10 trajectories for training, and 12 environments with 10 trajectories for testing.

We use the solver from Brandstetter et al. (2022d), and take 250 snapshots along the 4s of the generations. We employ a spatial resolution of 256 and downsample the temporal resolution to 25 frames.

C.2.4 Wave boundary

We consider a 1D wave equation as in Brandstetter et al. (2022d).

$$\partial_{tt} u - c^2 \partial_{xx} u = 0, \quad x \in [-8, 8]$$

where c is the wave velocity ($c = 2$ in our experiments). We consider Dirichlet $\mathcal{B}[u] = u = 0$ and Neumann $\mathcal{B}[u] = \partial_x u = 0$ boundary conditions.

We consider 4 different environments as each boundary can either respect Neumann or Dirichlet conditions, and sample 3000 trajectories for each environment. This results in 12000 trajectories for training. For the test set, we sample 30 new trajectories from these 4 environments resulting in 120 test trajectories.

The initial condition is a Gaussian pulse with a peak at a random location. Numerical ground truth is generated with the solver proposed in Brandstetter et al. (2022d). We obtain ground truth trajectories with resolution $(n_x, n_t) = (256, 250)$, and downsample the temporal resolution to obtain trajectories of shape $(256, 60)$.

C.2.5 Combined equation

We used the setting introduced in Brandstetter et al. (2022d), but with the exception that we do not include a forcing term. The combined equation is thus described by the following PDE:

$$[\partial_t u + \partial_x(\alpha u^2 - \beta \partial_x u + \gamma \partial_{xx} u)](t, x) = \delta(t, x), \tag{C.1}$$

$$\delta(t, x) = 0, \quad u_0(x) = \sum_{j=1}^{J} A_j \sin(2\pi \ell_j x / L + \phi_j). \tag{C.2}$$

For training, we sampled 1200 triplets of parameters uniformly within the ranges $\alpha \in [0, 1]$, $\beta \in [0, 0.4]$, and $\gamma \in [0, 1]$. For each parameter instance, we sample 10 trajectories, resulting in 12000 trajectories for training and 120 trajectories for testing. We used the solver proposed in Brandstetter et al. (2022c) to generate the solutions. The trajectories were generated with a spatial resolution of 256 for 10 seconds, along which 140 snapshots are taken. We downsample the temporal resolution to obtain trajectories with shape (256, 14).

C.2.6 Vorticity

We propose a 2D turbulence equation. We focus on analyzing the dynamics of the vorticity variable. The vorticity, denoted by ω, is a vector field that characterizes the local rotation of fluid elements, defined as $\omega = \nabla \times \mathbf{u}$. The vorticity equation is expressed as:

$$\frac{\partial \omega}{\partial t} + (\mathbf{u} \cdot \nabla)\omega - \nu \nabla^2 \omega = 0 \tag{C.3}$$

Here, $\mathbf{u}$ represents the fluid velocity field, ν is the kinematic viscosity with $\nu = 1/Re$. For the vorticity equation, the parametric problem consists in learning dynamical systems with changes in the viscosity term.

For training, we sampled 1200 PDE parameter values in the range $\nu = [1e-3, 1e-2]$. For test, we evaluate our model on 120 new parameters not seen during training in the same paramter range. For each parameter instance, we sample 10 trajectory, resulting in 12000 trajectories for training and 1200 for test.

Data generation For the data generation, we use a 5 point stencil for the classical central difference scheme of the Laplacian operator. For the Jacobian, we use a second order accurate scheme proposed by Arakawa that preserves the energy, enstrophy and skew symmetry (Arakawa, 1966). Finally for solving the Poisson equation, we use a Fast Fourier Transform based solver. We discretize a periodic domain into 512×512 points for the DNS and uses a RK4 solver with $\Delta t = 1\text{e}-3$ on a temporal horizon $[0, 2]$. We then perform a temporal and spatial down-sample operation, thus obtaining trajectories composed of 10 states on a 64×64 grid.

We consider the following initial conditions:

$$E(k) = \frac{4}{3}\sqrt{\pi}\left(\frac{k}{k_0}\right)^4 \frac{1}{k_0} \exp\left(-\left(\frac{k}{k_0}\right)^2\right) \tag{C.4}$$

Vorticity is linked to energy by the following equation :

$$\omega(k) = \sqrt{\frac{E(k)}{\pi k}} \tag{C.5}$$

C.2.7 Wave 2D

We propose a 2D damped wave equation, defined by

$$\frac{\partial^2 \omega}{\partial t^2} - c^2 \Delta \omega + k \frac{\partial \omega}{\partial t} = 0 \tag{C.6}$$

where c is the wave speed and k is the damping coefficient. We are only interested in learning ω. To tackle the parametric problem, we sample 1200 parameters in

the range $c = [0, 50]$ and $k = [100, 500]$. For validation, we evaluate our model on 120 new parameters not seen during training in the same paramter range. For each parameter instance, we sample 10 trajectory, resulting in 12000 trajectories for training and 1200 for validation.

Data generation For the data generation, we consider a compact spatial domain Ω represented as a 64×64 grid and discretize the Laplacian operator similarly. Δ is implemented using a 5×5 discrete Laplace operator in simulation. For boundary conditions, null neumann boundary conditions are imposed. We set $\Delta t = 6.25e - 6$ and generate trajectories on the temporal horizon $[0, 5e - 3]$. The simulation was integrated using a fourth order runge-kutta schema from an initial condition corresponding to a sum of gaussians:

$$\omega_0(x, y) = C \sum_{i=1}^{p} \exp\left(-\frac{(x - x_i)^2 + (y - y_i)^2}{2\sigma_i^2}\right) \tag{C.7}$$

where we choose $p = 5$ gaussians with $\sigma_i \sim \mathcal{U}_{[0.025,0.1]}$, $x_i \sim \mathcal{U}_{[0,1]}$, $y_i \sim \mathcal{U}_{[0.,1]}$. We fixed C to 1 here. Thus, all initial conditions correspond to a sum of gaussians of varying amplitudes.

C.3 Architecture details

C.3.1 Baseline implementations

For all baselines, we followed the recommendations given by the authors. We report here the architectures used for each baseline:

- CODA: For CODA, we implemented a U-Net Ronneberger et al. (2015a) and a FNO (Li et al., 2020a) as the neural network decoder. For all the different experiments, we reported in the results the best score among the two backbones used. We trained the different models in the same manner as Zebra, i.e. via teacher forcing (Radford et al., 2018). The model is adapted to each environment using a context vector specific to each environment. For the size of the context vector, we followed the authors recommendation and chose a context size equals to the number of degrees of freedom used to define each environment for each dataset. At inference, we adapt to a new environment using 250 gradient steps.

- CAPE: For CAPE (Takamoto et al., 2023), we adapted the method to an adaptation setting. Instead of giving true physical coefficients as input, we learn to auto-decode a context vector c^e as in CODA, which implicitly embeds the specific characteristics of each environment. During inference, we only adapt

c^e with 250 gradient steps. For the architectures, we use UNET and FNO as the backbones, and reported the best results among the two architectures for all settings.

- `[CLS]` ViT: For the ViT, we use a simple vision transformer architecture Dosovitskiy et al. (2021), but adapt it to a meta-learning setting where the CLS token encodes the specific variations of each environment. At inference, the CLS token is adapted to a new environment with 100 gradient steps.

- ViT`-in-context`: We implement ViT`-in-context` using a standard transformer architecture with separate temporal and spatial attention mechanisms, following Ho et al. (2019). During both training and inference, context examples are stacked along the temporal dimension. The model is trained to predict the next frame in the target sequence, conditioned on both the context examples and the preceding frames of the target sequence.

C.3.2 Zebra additional details

Zebra We describe the pretraining strategy in Section 8.3, and provide details on the architecture and its hyperparameters in Appendix C.3. The datasets used are described in Appendix C.2. We plan to release the code, the weights of the models, and the datasets used in this study upon acceptance.

For clarity, we outline the pretraining steps of `Zebra` in Figure C.1 illustrated with the *vorticity 2D* dataset.

We also provide illustrations of our inference pipeline in Figure C.2. We finally include a schematic view of the different generation possibilities with `Zebra` in Figure C.4, using the sequence design adopted during pretraining.

Zebra + UNet `Zebra` is competitive both in-distribution and out-of-distribution, while also enabling uncertainty quantification due to its generative nature. Additionally, `Zebra` can generate novel trajectories and initial conditions, providing a way to sample complex initial states. As such Zebra is already faster than gradient-based adaptation methods, but since the model generates trajectory solutions token by token, the number of calls to the transformer increases by one or two orders of magnitude compared to direct surrogate modeling, making the process costly.

In this section, we propose a hybrid approach that leverages `Zebra`'s pretrained knowledge in combination with a conventional neural surrogate model. The objective is to develop a framework that encodes context trajectories into an embedding vector, which then conditions a neural surrogate, similar to classical adaptation methods such as CODA and CAPE.

To achieve this, we finetune `Zebra` as an encoder to adapt a conditional surrogate model, such as a UNet (Ronneberger et al., 2015a). We introduce a `[DYN]` token

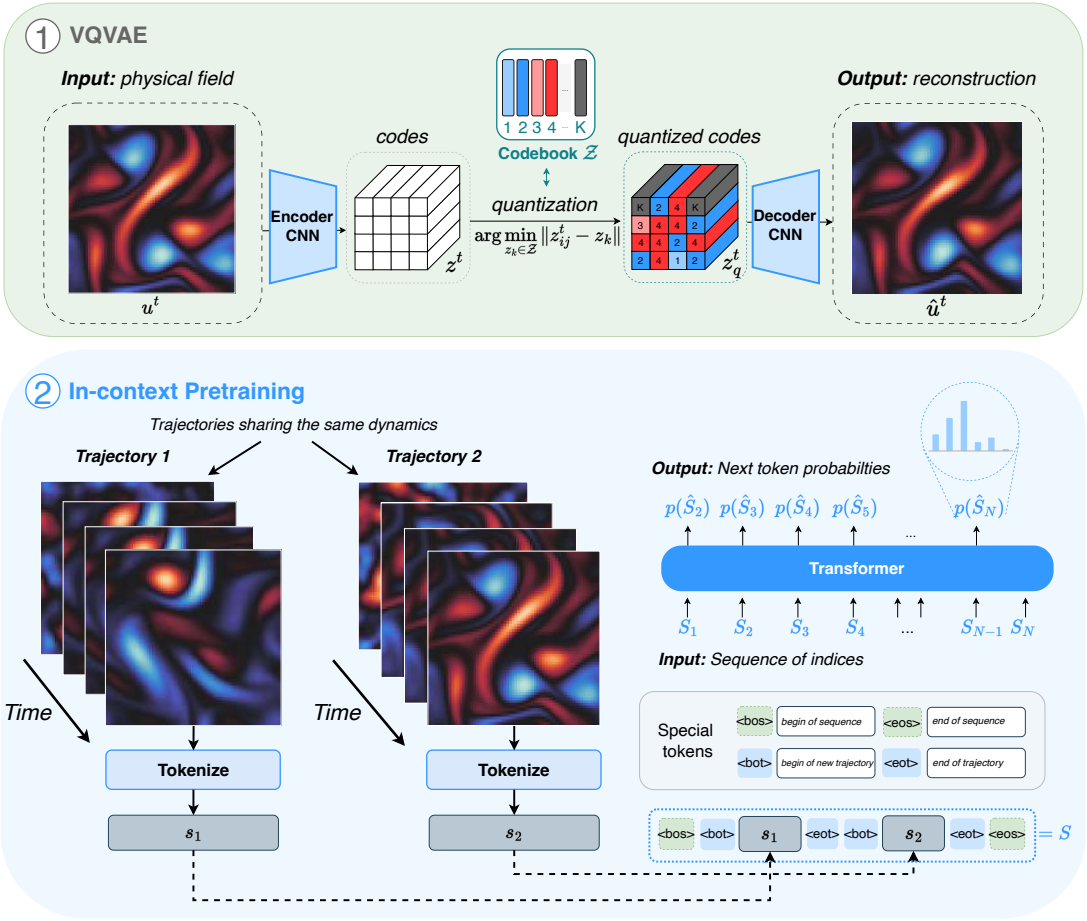


Figure C.1: `Zebra`'s pretraining includes two stages. 1) A finite vocabulary of physical phenomena is learned by training a VQ-VAE on spatial representations. 2) During the pretraining, multiple trajectories sharing the same dynamics are tokenized and concatenated into a common sequence S. The transformer is trained to predict the next-token by minimizing the cross-entropy loss.

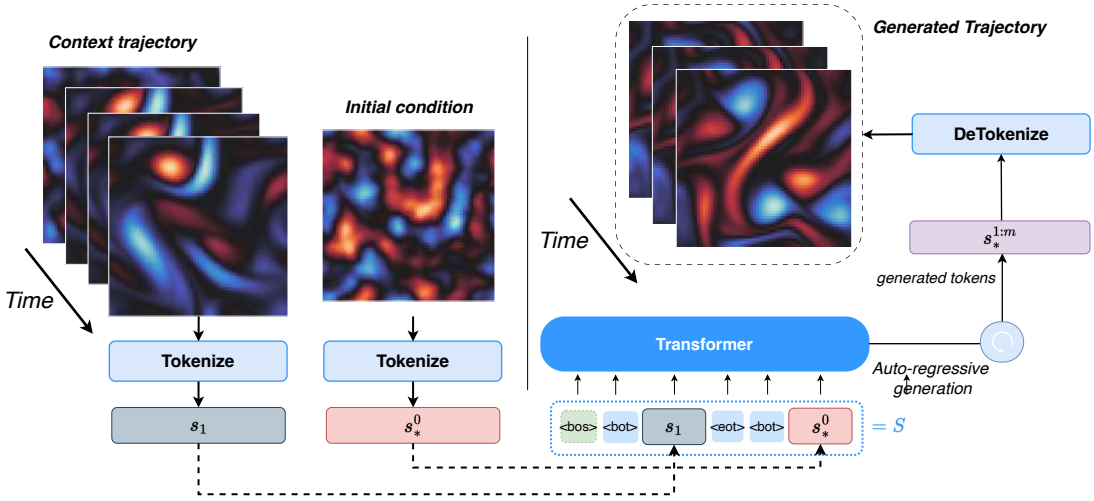


Figure C.2: `Zebra`'s inference pipeline from context trajectory. The context trajectory and initial conditions are tokenized into index sequences that are concatenated according to the sequence design adopted during pretraining. The transformer then generates the next tokens to complete the sequence. We detokenize these indices to get back to the physical space.

(short for dynamics), which is appended to the right of the context sequence during both training and inference. This allows `Zebra` to extract a dynamics embedding from the transformer's output, defined as:

$$\xi_S = \text{Transformer}(S, \texttt{[DYN]})$$

where S represents a sequence of tokens encoding the context trajectories. The embedding ξ_S captures key properties of the dynamics and is mapped to the conditioning space of a UNet, which adapts the model to each specific dynamics. Following Gupta and Brandstetter (2022), the UNet conditioning modifies the network's biases. Once the dynamics embedding is extracted, the UNet can directly predict the next state from the current state :

$$\hat{\boldsymbol{u}}^{t+\Delta t} = \text{UNet}(\boldsymbol{u}^t, \xi_S).$$

With this architecture, we effectively extract the key dynamics from example trajectories and use this information to adapt a neural surrogate, significantly accelerating inference. The complete pipeline at inference is illustrated in Figure C.3.

To efficiently finetune `Zebra`, we apply LoRA (Hu et al., 2021), keeping the transformer's weights frozen while learning low-rank updates. This setup enables the UNet to leverage `Zebra`'s pre-learned representations while achieving a substantial speedup. Compared to standalone `Zebra`, integrating a UNet improves inference speed by a factor of $\times 30$ in 1D and $\times 150$ in 2D. The method is also considerably

faster than CODA and CAPE while maintaining competitive performance across tasks. The inference times are summarized in Table 8.4.

Finally we illustrate the inference pipeline for accelerating the inference of `Zebra` with the UNet in Figure C.3.

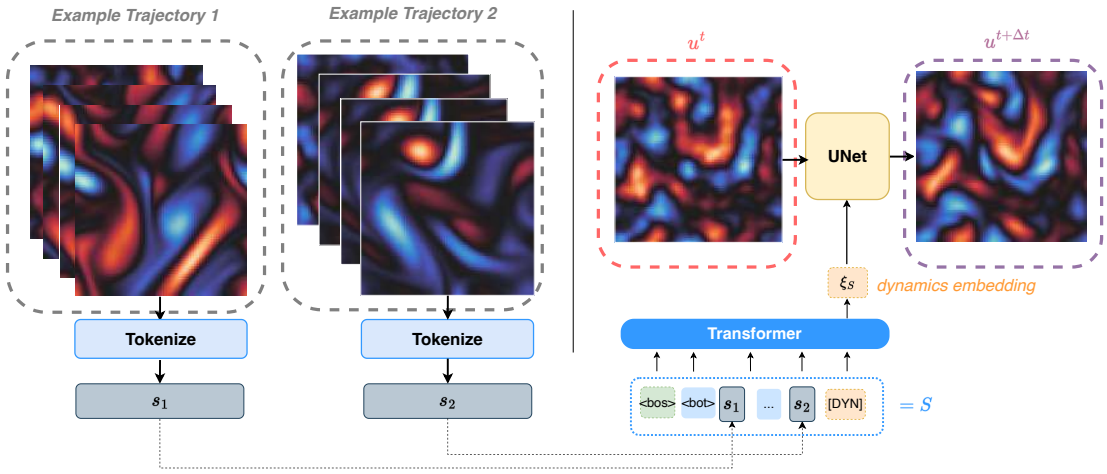


Figure C.3: `Zebra` + UNet inference pipeline. `Zebra` serves as an encoder, utilizing the special token `[DYN]` to generate a dynamics embedding ξ from the context example trajectories. Once this embedding is obtained, the UNet can autoregressively forecast the sequence significantly faster than a next-token transformer.

C.3.3 Auto-regressive transformer

`Zebra`'s transformer is based on Llama's architecture, which we describe informally in Figure C.5. We use the implementation provided by HuggingFace (Wolf, 2019) and the hyperparameters from Table C.2 in our experiments. For training the transformer, we used a single NVIDIA TITAN RTX for the 1D experiments and used a single A100 for training the model on the 2D datasets. Training the transformer on 2D datasets took 20h on a single A100 and it took 15h on a single RTX for the 1d dataset.

C.3.4 VQVAE

The quantizer used at the token level is a VQVAE model (Oord et al., 2017). As illustrated in Figure C.6 this is an encoder-decoder architecture with an intermediate quantizer component.

The encoder spatially compresses the input function $\boldsymbol{u}^t$ by reducing its spatial resolution $H \times W$ to a lower resolution $h \times w$ while increasing the channel dimension to d. This is achieved through a convolutional model $\mathcal{E}_w$, which maps the input to a continuous latent variable $\mathbf{z}^t = \mathcal{E}_w(\boldsymbol{u}^t)$, where $\mathbf{z}^t \in \mathbb{R}^{h \times w \times d}$. The latent variables

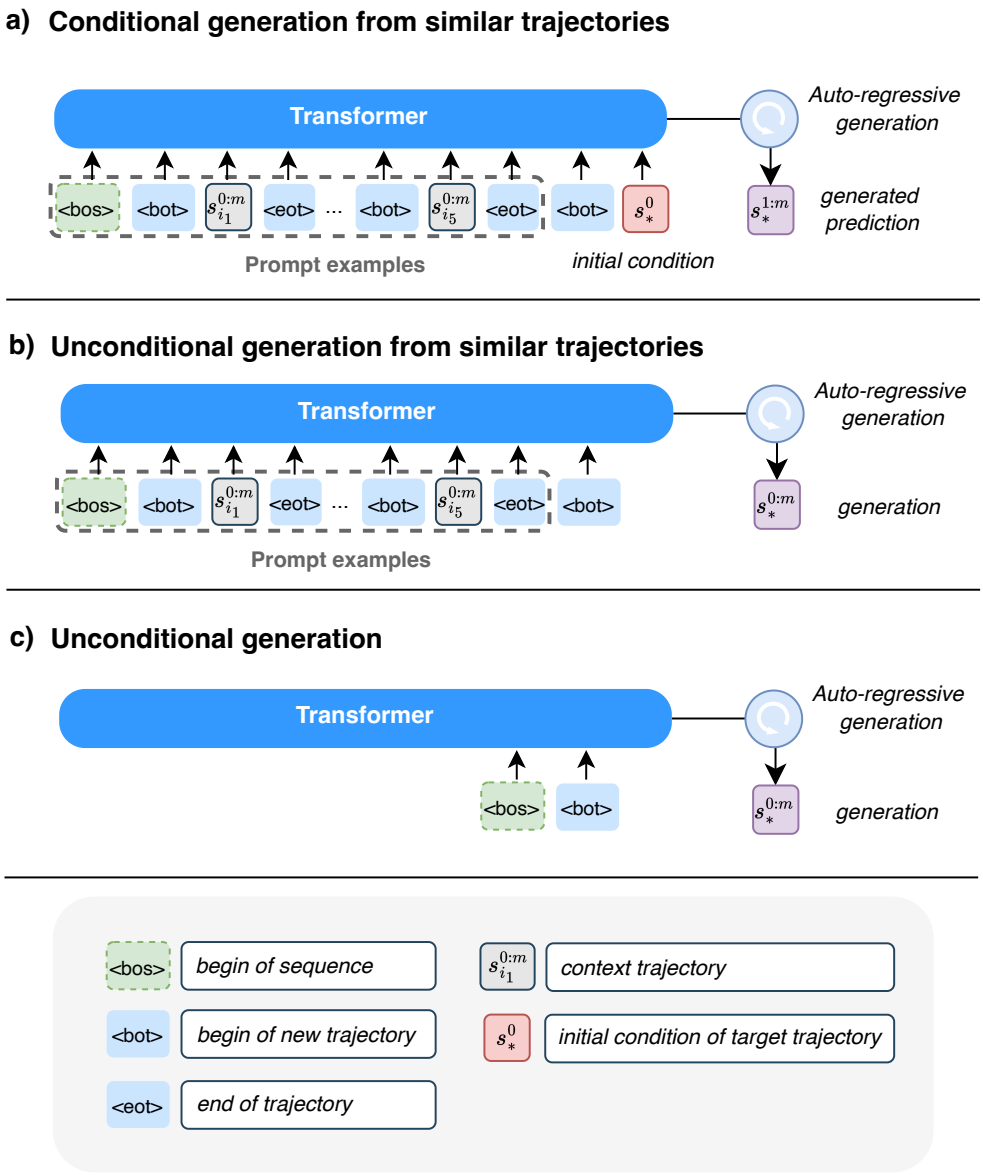


Figure C.4: Generation possibilities with `Zebra`.

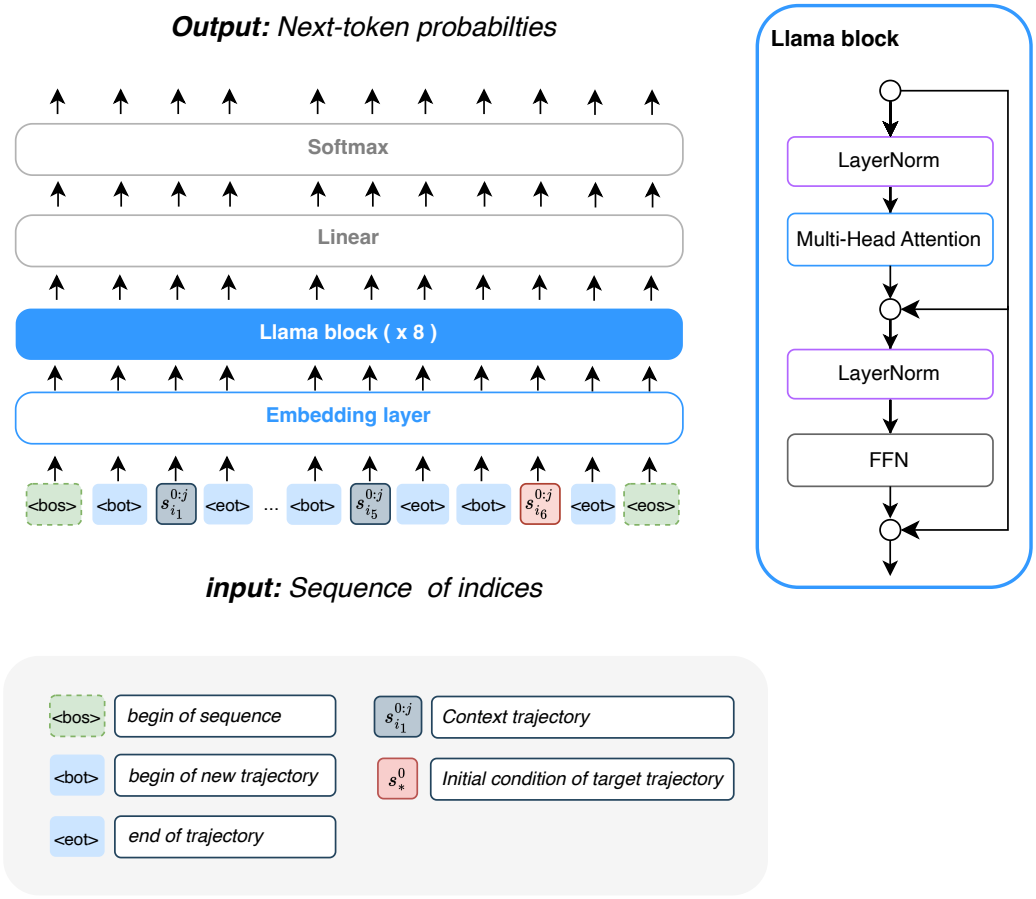


Figure C.5: `Zebra`'s transformer architecture is based on Llama (Touvron et al., 2023).

Table C.2: Hyperparameters for Zebra's Transformer

Hyperparameters	Advection	Heat	Burgers	Wave b	Combined	Vorticity 2D	Wave 2D
max_context_size	2048	2048	2048	2048	2048	8192	8192
batch_size	4	4	4	4	4	2	2
num_gradient_accumulations	1	1	1	1	1	4	4
hidden_size	256	256	256	256	256	384	512
mlp_ratio	4.0	4.0	4.0	4.0	4.0	4.0	4.0
depth	8	8	8	8	8	8	8
num_heads	8	8	8	8	8	8	8
vocabulary_size	264	264	264	264	264	2056	2056
start learning_rate	1e-4	1e-4	1e-4	1e-4	1e-4	1e-4	1e-4
weight_decay	1e-4	1e-4	1e-4	1e-4	1e-4	1e-4	1e-4
scheduler	Cosine	Cosine	Cosine	Cosine	Cosine	Cosine	Cosine
num_epochs	100	100	100	100	100	30	30

are then quantized to discrete codes $\mathbf{z}_q^t$ using a codebook $\mathcal{Z}$ of size $K = |\mathcal{Z}|$ and through the quantization step q. For each spatial code $\mathbf{z}_{[ij]}^t$, the nearest codebook entry z_k is selected:

$$\mathbf{z}_{q,[ij]}^t = q(\mathbf{z}_{[ij]}^t) := \arg\min_{z_k \in \mathcal{Z}} \|\mathbf{z}_{[ij]}^t - z_k\|.$$

The decoder $\mathcal{D}_\psi$ reconstructs the signal $\hat{\boldsymbol{u}}^t$ from the quantized latent codes $\hat{\mathbf{z}}_q^t$. Both models are jointly trained to minimize the reconstruction error between the function $\boldsymbol{u}^t$ and its reconstruction $\hat{\boldsymbol{u}}^t = \mathcal{D}_\psi \circ q \circ \mathcal{E}_w(\boldsymbol{u}^t)$. The codebook $\mathcal{Z}$ is updated using an exponential moving average (EMA) strategy, which stabilizes training and ensures high codebook occupancy.

The training objective is:

$$\mathcal{L}_{\text{VQ}} = \frac{\|\boldsymbol{u}^t - \hat{\boldsymbol{u}}^t\|_2}{\|\boldsymbol{u}^t\|_2} + \alpha \|\text{sg}[\mathbf{z}_q^t] - \mathcal{E}_w(\boldsymbol{u}^t)\|_2^2,$$

where the first term is the Relative L2 loss commonly used in PDE modeling, and the second term is the commitment loss, ensuring encoder outputs are close to the codebook entries. The parameter α, set to 0.25, balances the two components. Here, sg denotes the stop-gradient operation that detaches a tensor from the computational graph.

We provide a schematic view of the VQVAE framework in Figure C.6 and detail the architectures used for the encoder and decoder on the 1D and 2D datasets respectively in Figure C.7 and Figure C.8. As detailed, we use residual blocks to process latent representations, and downsampling and upsampling block for decreasing / increasing the spatial resolutions. We provide the full details of the hyperparameters used during the experiments in Table C.3. For training the VQVAE, we used a single NVIDIA TITAN RTX for the 1D experiments and used a single V100 for training the model on the 2D datasets. Training the encoder-decoder on 2D datasets took 20h on a single V100 and it took 4h on a single RTX for 1D dataset.

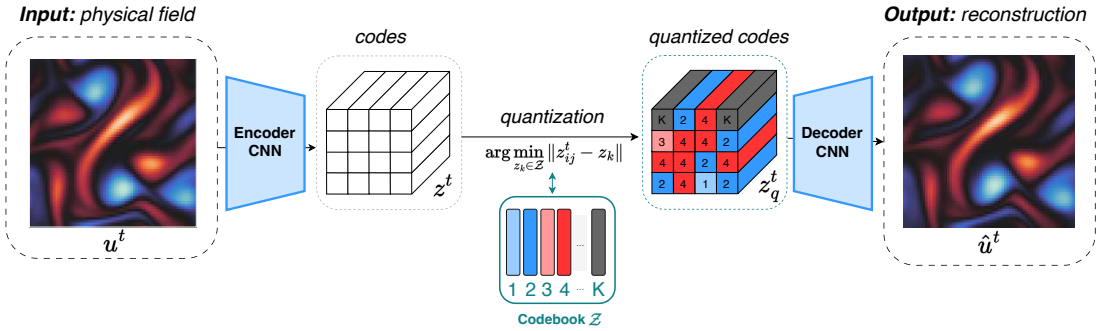


Figure C.6: `Zebra`'s VQVAE is used to obtain compressed and discretized latent representation. By retrieving the codebok index for each discrete representation, we can obtain discrete tokens encoding physical observations that can be mapped back to the physical space with high fidelity.

Table C.3: Hyperparameters for Zebra's VQVAE

Hyperparameters	Advection	Heat	Burgers	Wave b	Combined	Vorticity 2D	Wave 2D
start_hidden_size	64	64	64	64	64	128	128
max_hidden_size	256	256	256	256	256	1024	1024
num_down_blocks	4	4	4	4	4	2	3
codebook_size	256	256	256	256	256	2048	2048
code_dim	64	64	64	64	64	16	16
num_codebooks	2	2	2	2	2	1	2
shared_codebook	True	True	True	True	True	True	True
tokens_per_frame	32	32	32	32	32	256	128
start learning_rate	3e-4	3e-4	3e-4	3e-4	3e-4	3e-4	3e-4
weight_decay	1e-4	1e-4	1e-4	1e-4	1e-4	1e-4	1e-4
scheduler	Cosine	Cosine	Cosine	Cosine	Cosine	Cosine	Cosine
num_epochs	1000	1000	1000	1000	1000	300	300

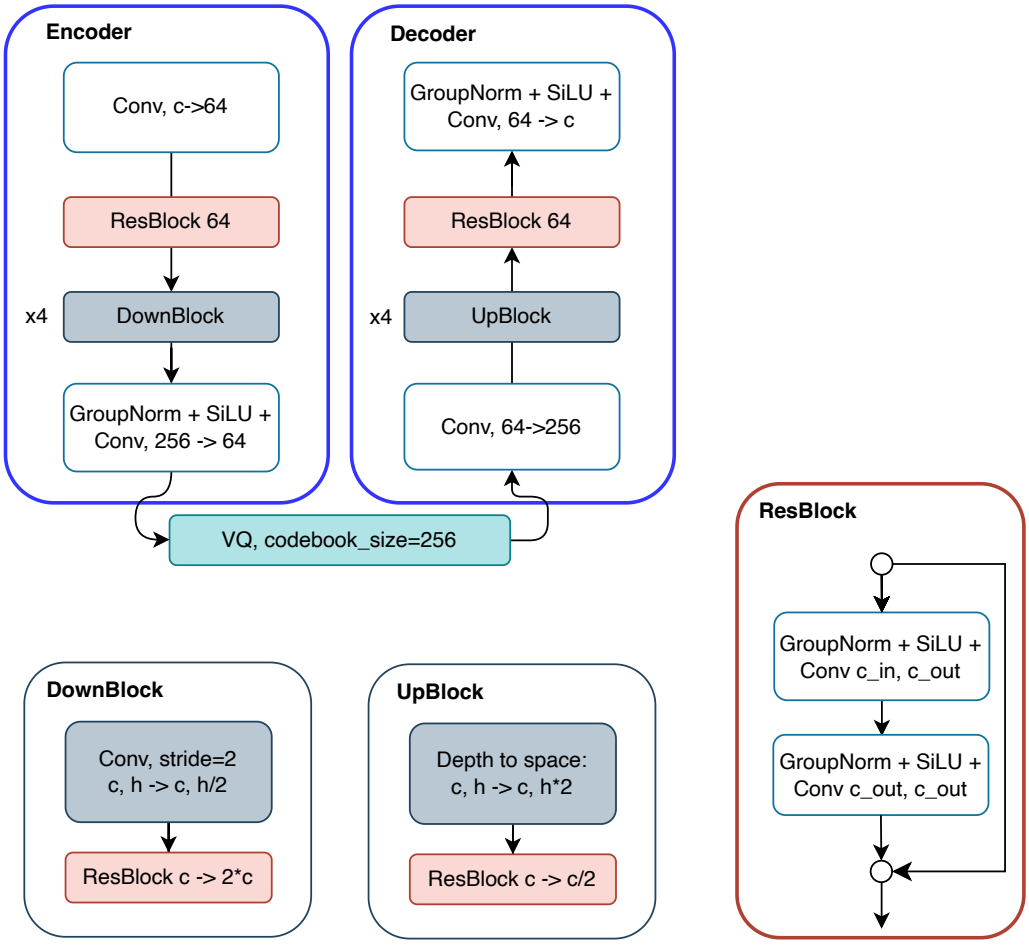


Figure C.7: Architecture of `Zebra`'s VQVAE for 1D datasets. Each convolution acts only on the spatial dimension and uses a kernel of size 3. The Residual Blocks are used to process information and increase or decrease the channel dimensions, while the Up and Down blocks respectively up-sample and down-sample the resolution of the inputs. In 1D, we used a spatial compression factor of 16 on all datasets. Every downsampling results in a doubling of the number of channels, and likewise, every upsampling is followed by a reduction of the number of channels by 2. We choose a maximum number of channels of 256.

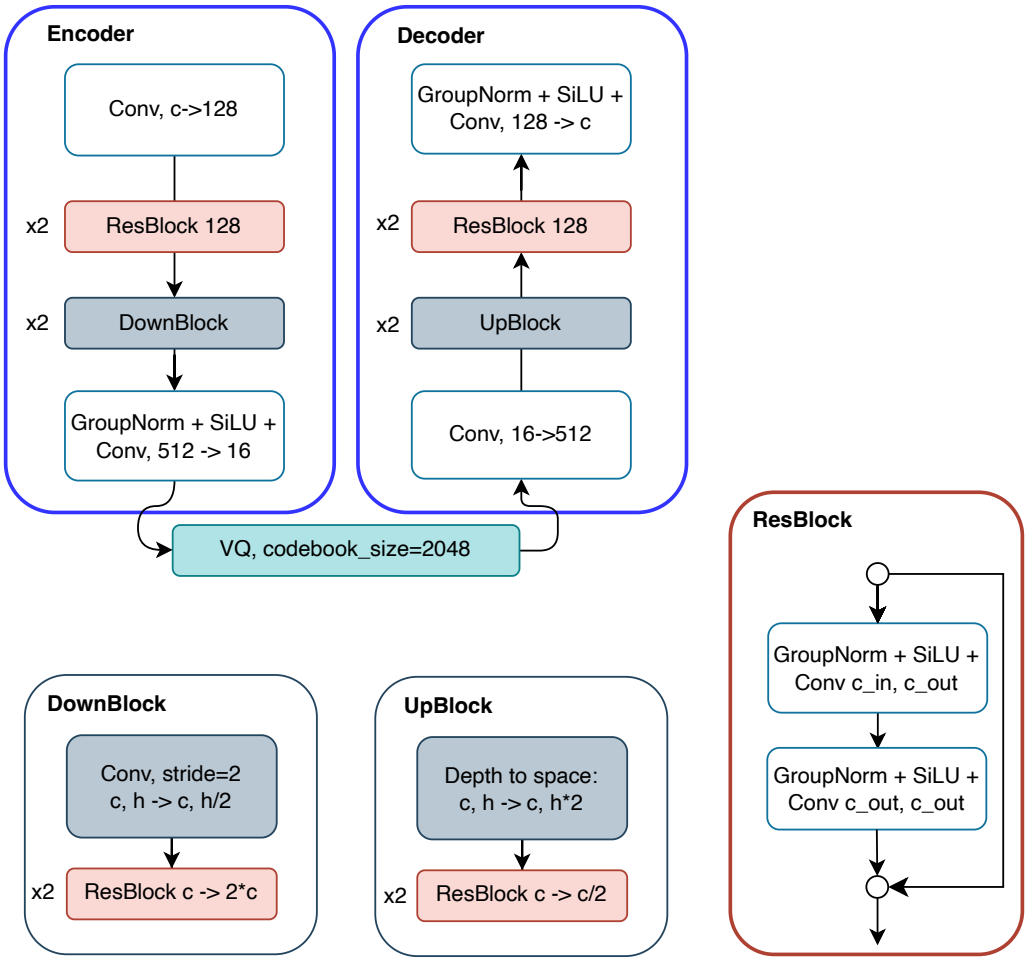


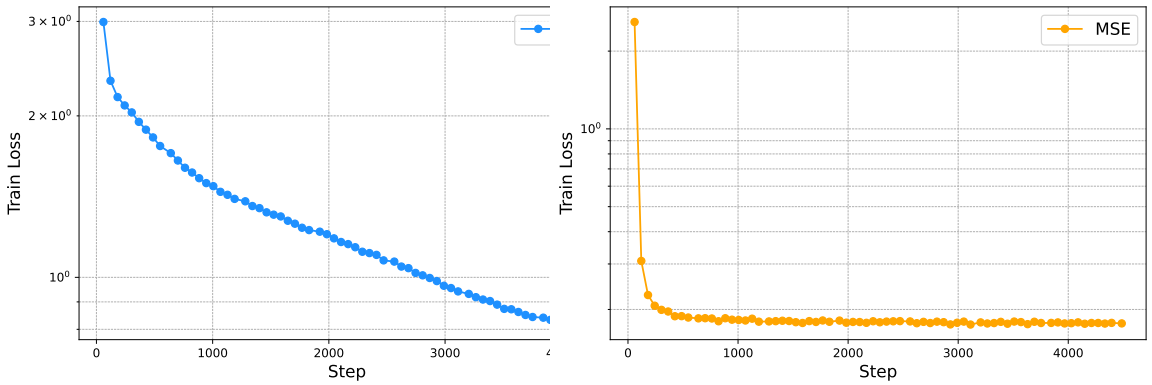
Figure C.8: Architecture of `Zebra`'s VQVAE for 2D datasets. Each convolution acts only on the spatial dimensions and uses a kernel of size 3. The Residual Blocks are used to process information and increase or decrease the channel dimensions, while the Up and Down blocks respectively up-sample and down-sample the resolution of the inputs. In 2D, we used a spatial compression factor of 4 for *Vorticity*, and 8 for *Wave2D*. Every downsampling results in a doubling of the number of channels, and likewise, every upsampling is followed by a reduction of the number of channels by 2. We choose a maximum number of channels of 1024.

C.4 Additional Quantitative results

C.4.1 Alternative pretrainings

We experimented with different pretraining strategies before settling on the pretraining approach proposed in `Zebra`. The most intuitive way to adapt the next-token prediction objective for dynamics modeling is to operate in a continuous latent space, omitting the quantization step used in `Zebra` and therefore using a deterministic transformer instead of a generative model. As shown in previous studies (Li et al., 2024b; Agarwal et al., 2025), we obtained better reconstruction results using an autoencoder instead of a VQVAE.

However, we encountered two critical challenges with this approach. First, the training loss plateaued quickly, as illustrated in Figure C.9. Compared to training a generative transformer (with the negative log-likelihood) as in `Zebra`, the deterministic variant trained with MSE loss exhibited instability and failed to improve over training steps.



Training loss (negative log-likelihood) with `Zebra`

Training loss (MSE) with deterministic transformer

Figure C.9: Comparison of the training process between the `Zebra` transformer and a deterministic transformer on *Advection*. In both cases, the model is trained to predict the next token. `Zebra` utilizes a discrete vocabulary and learns a probability distribution over the next token, whereas the deterministic transformer is optimized to predict the mean using MSE loss.

Second, while the model was able to predict the next token at inference, it could not generate an entire trajectory. It performed particularly poorly in the one-shot adaptation setting. As demonstrated in Figure C.10, errors accumulated quickly during inference, leading to rapid divergence from the ground truth. This ultimately resulted in poor reconstructions when feeding the predicted tokens into the decoder, as seen in Figure C.11.

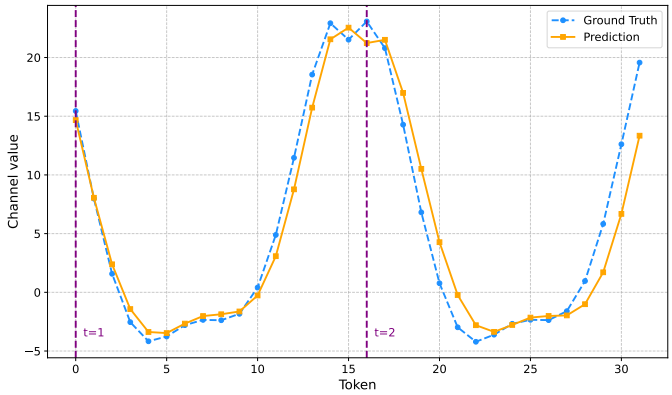


Figure C.10: When trained with MSE for next-token prediction, inference suffers from instabilities, causing errors to grow exponentially. Here, we show the evolution over the sequence tokens of a particular channel. The dashed lines show the temporal transitions between two subsequent frames.

These observations highlight the importance of the generative aspect when adopting a next-token objective. For this reason, we opted for a quantized representation combined with a transformer modeling a discrete distribution, a standard approach in image and video generation, but which has never been explored for modeling physical phenomena. While alternative strategies exist (Tian et al., 2024), they involve non-trivial extensions and are left for future works.

That said, next-token prediction pretraining may not be the only viable framework for developing in-context capabilities. To explore this, we experimented with a direct next-frame prediction approach using a deterministic setup trained with relative L2 loss. This method, which we called `VIT-in-context` serves as a baseline for evaluating other in-context pretrainings. It is based on a video transformer operating on patches with bidirectional attention. While its training behavior was more stable, inference results remained unsatisfactory.

C.4.2 Uncertainty quantification

Setting Since `Zebra` is a generative model, it allows us to sample multiple plausible trajectories for the same conditioning input, enabling the computation of key statistics across different generations. By calculating the pointwise mean and standard deviation, we can effectively visualize the model's uncertainty in its predictions. In Figure C.12, the red curve represents the ground truth, the blue curve is the predicted mean and the blue shading indicates the empirical confidence interval ($3 \times$ standard deviation).

Metrics Motivated by this observation, we investigate how varying the model's temperature parameter τ affects its predictions; specifically in the one-shot adapta-

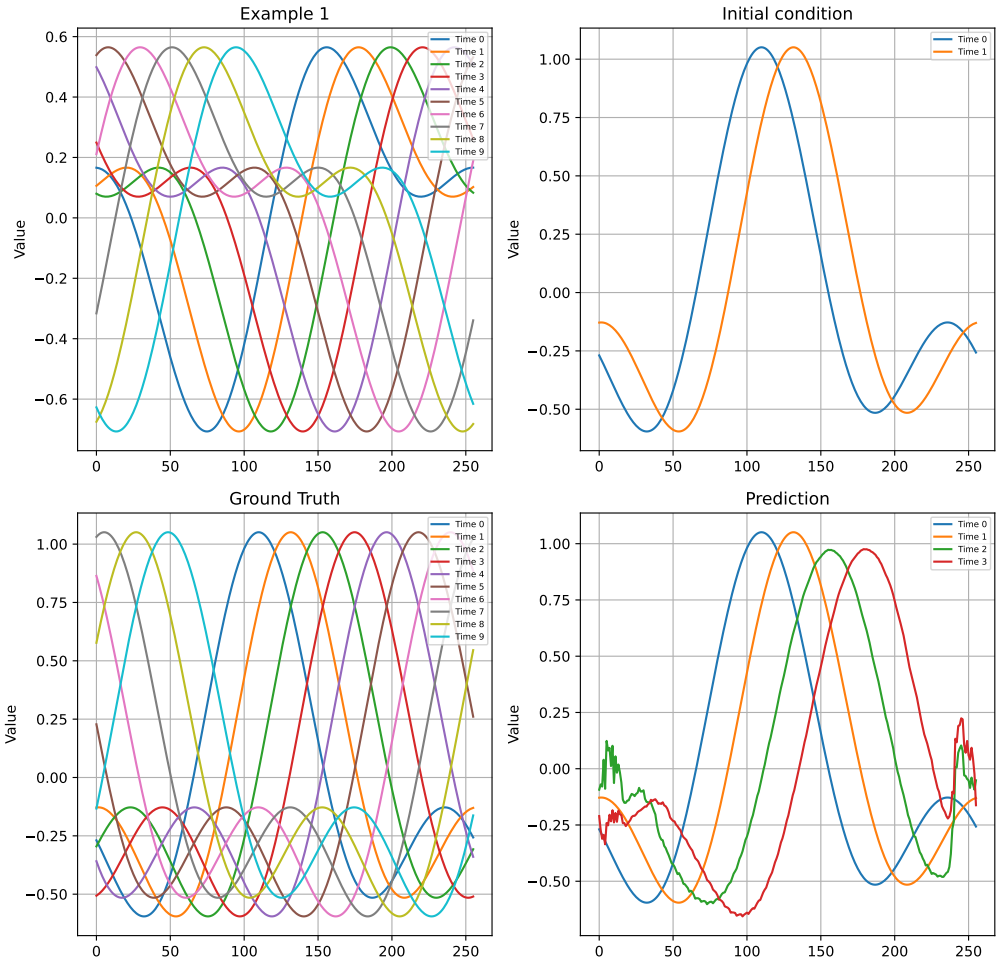


Figure C.11: The deterministic transformer, trained with the next-token objective, performs poorly in the one-shot adaptation task.

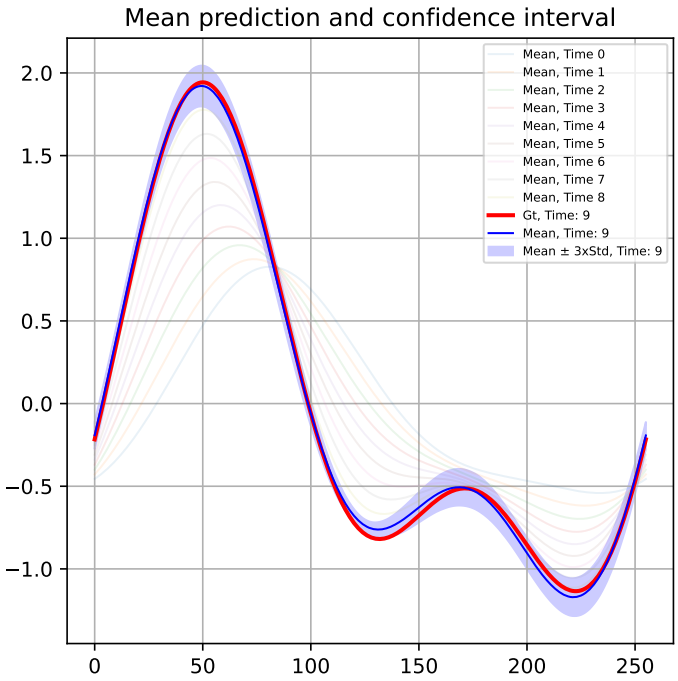


Figure C.12: Uncertainty quantification with `Zebra` in a one-shot setting on *Heat* equation

tion setting described in Section 8.4.2. By adjusting τ, we aim to assess its impact on both the accuracy and variability of the predictions. We generate 10 different trajectories and use them to compute several metrics. We employ four metrics to evaluate the model's uncertainty:

1. **Relative L^2 loss**: This assesses the accuracy of the generated trajectories by measuring the bias of the predictions relative to the ground truth.

2. **Relative standard deviation**: We estimate the variability of the predictions using the formula: Relative Std $= \frac{||\hat{\sigma}_*||_2}{||\hat{m}_*||_2}$ where $\hat{m}_*$ and $\hat{\sigma}_*$ represent the empirical mean and standard deviation of the predictions, computed pointwise across 10 generations.

3. **Confidence level**: We create pointwise empirical confidence intervals $CI(x) = [\hat{m}_*(x) - 3\hat{\sigma}_*(x), \hat{m}_*(x) + 3\hat{\sigma}_*(x)]$ and compute the confidence level as: Confidence level $= \frac{1}{n_x}\sum_x \mathbf{1}_{u_*(x)\in CI(x)}$. This score indicates how often the ground truth falls within the empirical confidence interval generated from sampling multiple trajectories.

4. **Continuous Ranked Probability Score** (CRPS, Gneiting and Raftery (2007)) is a proper scoring rule that measures the accuracy of a probabilistic

forecast by quantifying the difference between the predicted cumulative distribution function (CDF) and the empirical CDF of the observed value, with lower values indicating better calibration and sharpness.

Results When modeling uncertainty, the model achieves a tradeoff between the quality of the mean prediction approximation and the guarantee for this prediction to be in the corresponding confidence interval. Figure C.13 illustrates the trade-off between mean prediction accuracy and uncertainty calibration. At lower temperatures, we achieve the most accurate predictions, but with lower variance, i.e. with no guarantee that the target value is within the confidence interval around the predicted mean. Across most datasets, the confidence level then remains low (less than 80% for $\tau < 0.25$), indicating that the true solutions are not reliably captured within the empirical confidence intervals. Conversely, increasing the temperature results in less accurate mean predictions and higher relative standard deviations, but the confidence intervals become more reliable, with levels exceeding 95% for $\tau > 0.5$. Therefore, the temperature can be calibrated depending on whether the focus is on accurate point estimates or reliable uncertainty bounds.

To better calibrate this temperature, we can therefore use a proper scroring metric such as the CRPS, and we can pick the temperature parameter that has the lowest CRPS value for a given value (see Figure C.13). We can see that Zebra models really well the distribution for Combined and Advection and not so much for Burgers and Heat somehow.

Finally, we examine how the model's uncertainty evolves as additional information is provided as input. Specifically, we compare Zebra's average error and relative uncertainty when conditioned on one example trajectory, with one or two frames as initial conditions. Table C.4 reports the relative L2 loss and relative standard deviation for both scenarios. The results clearly show that including the first two frames as initial conditions reduces both the error and the relative standard deviation consistently. This indicates that, while some of the uncertainty remains aleatoric, the epistemic uncertainty is reduced as more input information becomes available.

Table C.4: Uncertainty quantification in the one-shot setting. Conditioning from a trajectory example and 1 frame or 2 frames as initial conditions. Metrics include relative L^2 loss (average accuracy) and relative standard deviation (average spread around the average prediction). The temperature is fixed at 0.1.

		Advection	*Heat*	*Burgers*	*Wave b*	*Combined*
Rel. L^2	1 frame	0.006	0.156	0.115	0.154	0.008
Rel. L^2	2 frames	0.004	0.047	0.052	0.075	0.005
Rel. Std.	1 frame	0.003	0.062	0.048	0.074	0.005
Rel. Std.	2 frames	0.002	0.019	0.018	0.040	0.003

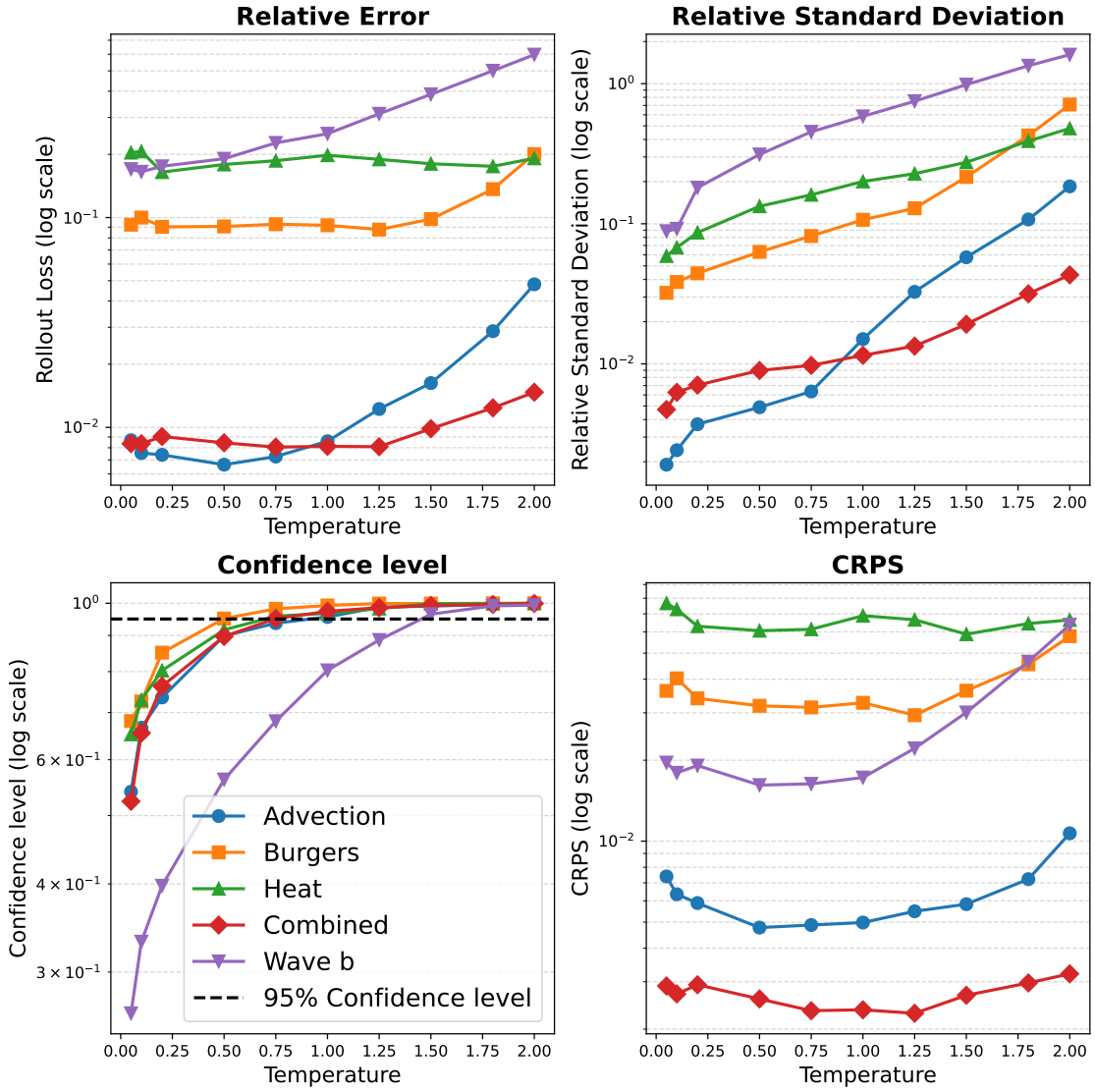


Figure C.13: Uncertainty quantification with `Zebra`. The main parameter of this study is the temperature (x-axis). We then look from left to right at (1) The rollout loss, i.e. the relative L^2 loss between the predictions and the ground truth; (2) The relative standard deviation to quantify the spread around the mean; (3) The confidence level, that measures the frequency of observations that lie within the empirical confidence interval. (4) The CRPS that measures the quality of the uncertainty, can be used to pick the temperature with the most calibrated uncertaity.

C.4.3 Analysis of the generation

`Zebra` is capable of generating completely novel trajectories for new environments, including the initial conditions. An example of a generated trajectory for *Vorticity 2D* is shown in Figure C.14, where the top row shows the context trajectory used to guide the generation, and the bottom row displays the model's generated trajectory, including the initial condition. In this section, we evaluate on *Combined* and *Advection* the quality of the generated trajectories with `Zebra`.

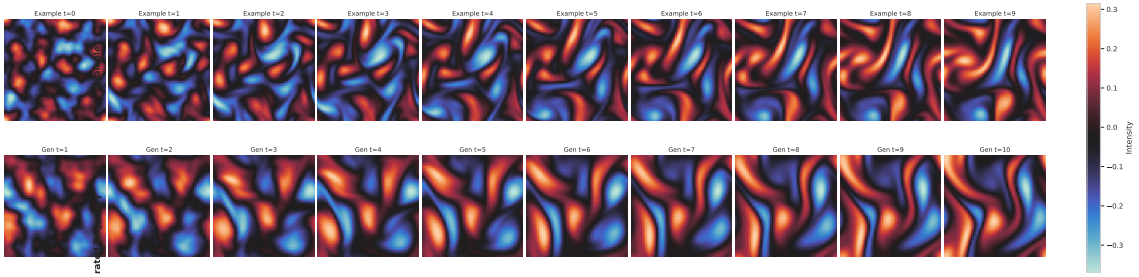


Figure C.14: Unconditional generation on *Vorticity 2D*. The top-row is the example used to guide the generation, and the bottom-row is the generated example. The model also generates the initial condition.

Setting We evaluate whether our pretrained model can generate new samples conditioned on a trajectory observed in a previously unseen test environment. Unlike previous settings, the transformer is not explicitly conditioned on tokens derived from a real initial condition. Instead, we expect it to generate trajectories, including their initial conditions, that follow the same dynamics as the observed context.

Our evaluation focuses on four key aspects. First, we assess whether the generated trajectories faithfully follow the dynamics of the context. Second, we analyze the diversity of the generated trajectories to determine if they are significantly different from one another. Third, we compare the generated samples with those produced by numerical solvers to evaluate whether their distributions align. Finally, we examine the types of initial conditions generated by `Zebra`.

Metrics To assess *fidelity*, we generate ground truth trajectories using the physical solver originally used to construct the dataset. These simulations start from the initial conditions generated by `Zebra`, using ground truth environment parameters that `Zebra` itself does not have access to. We then compute the L^2 distance between the generated trajectories and those obtained from the physical solver.

For *diversity*, we measure the average pairwise L^2 distance between different trajectories generated by `Zebra`. The results for both fidelity and diversity are reported in Table C.5 for the Advection and Combined Equations.

To compare the distribution of generated trajectories with that of numerical solvers, we compute the Wasserstein distance using the Sinkhorn algorithm Cu-

turi (2013). As baselines, we compare against two references. First, we compute the Wasserstein distance with purely random samples drawn from Gaussian noise, providing an upper bound on the problem. Second, we measure the Wasserstein distance between two independent sets of numerical solver trajectories (the validation and test sets), allowing us to quantify the variability inherent in the dataset itself.

Finally, as a qualitative analysis, we perform a principal component analysis on the trajectories generated by `Zebra` and visualize the first two principal components in Figure C.15 for *Combined*.

Sampling We use a default temperature of $\tau = 1.0$. For each context trajectory, we sample 10 new trajectories in parallel.

Results Table C.5 shows that `Zebra` generates new initial conditions and trajectories that respect the same physical laws as the given context. The model seems to have learned the statistical relationships between initial conditions and later timestamps. The high average L^2 distance between samples indicates that the generated trajectories are diverse. This can also be observed in Figure C.15, where the generated samples effectively cover the distribution of real samples.

Table C.5: Fidelity and diversity metrics. The L^2 distance measures fidelity to the context dynamics, while the average L^2 quantifies sample diversity.

Model	L^2	Average L^2 between samples
Advection	0.0185	1.57
Combined Equation	0.0136	1.59

To further assess distribution alignment, we compute the Wasserstein distance between the generated trajectories and those obtained with numerical solvers. The results in Table C.6 indicate that `Zebra` achieves lower Wasserstein distances than Gaussian noise but remains slightly above the cross-distribution baseline (which compares the validation and test distributions), suggesting reasonable alignment with the true data distribution.

Table C.6: Comparison of distributions using the Wasserstein distance between `Zebra`-generated trajectories and numerical solver samples.

Distance Metric	Advection	Combined Equation
Gaussian noise vs. real data	18.22	16.15
Validation data vs. test data	5.11	1.87
`Zebra`-generated data vs. real data	5.57	2.21

Distribution of generated initial conditions ($t = 0$).

Distribution of generated trajectories at ($t = 9$).

Figure C.15: Qualitative analysis of generated trajectories. `Zebra` generates new initial conditions and trajectories for unseen test environments. PCA projections visualize both generated and true trajectories in a lower-dimensional space at $t = 0$ and $t = 9$.

C.4.4 Dataset scaling analysis

We investigate how the one-shot error on the test set evolves as we vary the size of the training dataset. To this end, we train the auto-regressive transformer on datasets containing 10, 100, 1000, and 12,000 trajectories and evaluate Zebra's generations on the test set, starting with two frames as inputs. The training time is proportional to the dataset size: for example, the number of training steps for 1,000 trajectories is 10 times the number of steps for 100 trajectories. The results are presented in Figure C.16.

First, we observe that Zebra requires a substantial amount of data to generalize effectively to different parameter values, even within the training distribution. This aligns with findings in the literature that transformers, especially auto-regressive transformers, excel at scaling —performing well on very large datasets and for larger model architectures. However, for smaller datasets, this approach may not be the most efficient. We believe that Zebra's potential resides when applied to large amounts of data, making it an ideal candidate for scenarios involving large-scale training.

Second, for the *Combined equation*, we notice that performance plateaus between 100 and 1,000 trajectories. This may be due to insufficient training or a lack of diverse examples, as the *Combined equation* is more challenging compared to the *Advection equation*, whose performance follows a more log-linear trend. This suggests that additional data or targeted training strategies might be needed to achieve better generalization for more complex equations.

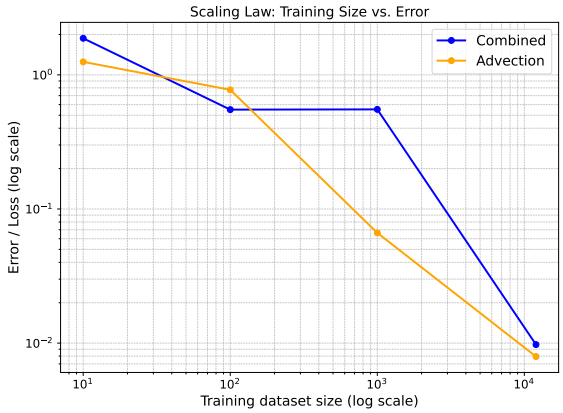


Figure C.16: Dataset scaling analysis. One-shot error on the test set vs. the training dataset size.

C.4.5 Inference time comparison

Table C.7 compares the inference time for one-shot adaptation across different methods when predicting a single trajectory given a context trajectory and an initial condition. For `Zebra`, the inference process, which includes encoding, auto-regressive prediction, and decoding, is much faster in 1D and slightly faster in 2D. With `Zebra`, the bottleneck at inference is the autoregressive generation of tokens, which speed is about 128 tokens per second on a V100 for 2D and an RTX for 1D. The decoding is fast and can be done in parallel for the trajectory in one forward pass. In contrast, for CODA and CAPE, the majority of the inference time is spent on adaptation and gradient-based steps. Here the times were reported with 100 gradient steps, note that we used 250 for the rest of the experiments. We believe Zebra's inference time could be further optimized by (1) improving the optimization code and leveraging specialized hardware such as H100 (for flash attention) and LPUs (which show significant speed-ups against GPUs), and (2) increasing the number of tokens sampled per step (as in e.g. next-scale prediction Tian et al. (2024)).

However, we have shown that it is possible to greatly accelerate inference by employing a deterministic neural surrogate on top of `Zebra`, which acts as a dynamic encoder. This framework is order of magnitudes faster than gradient-based adaptation methods.

C.4.6 Influence of the codebook size

The codebook size K is a crucial hyperparameter. It directly affects the quality of the reconstructions, since a larger codebook can improve the reconstructions quality. However, it also impacts the dynamics modeling stage: the smaller the codebook, the easier it is for the transformer to learn the statistical correlations between similar trajectories. To have a sense of this trade-off, we report the relative reconstruction

Table C.7: Inference times for one-shot adaptation. Average time in seconds to predict a single trajectory given a context trajectory and an initial condition. Times include adaptation and forecast for CODA and CAPE, while it includes encoding, auto-regressive prediction and decoding for Zebra.

	Advection	*Vorticity 2D*
CAPE	18s	23s
CODA	31s	28s
Zebra	3s	21s
Zebra + UNet	**0.10s**	**0.14s**

errors and the one-shot prediction errors in Table C.8. The reconstruction error decreases when the codebook size increases. However, the one-shot prediction error decreases from 32 to 64 codes but then gradually increases from 64 to 512. We can see that it follows a U-curve in Figure C.17. This phenomenon was observed in a different context in Cole et al. (2024).

Table C.8: Influence of the codebook size. Reconstruction error and one-shot prediction error on *Burgers* for different codebook sizes. Errors in relative L2.

Codebook Size	**Reconstruction Loss**	**One-shot Prediction**
32	0.0087	0.116
64	0.0043	0.097
128	0.0024	0.124
256	0.0019	0.163
512	0.0015	1.093

C.4.7 Reconstruction errors

We report the accuracy of the reconstructions from our encoder-decoder in Table C.9. Here, no dynamics is involved, we simply evaluate the quality of the encoding and of the decoding. On 1D and 2D datasets, the decoding errors are respectively of 0.1 % and 1% on the test set.

Table C.9: Reconstruction errors. Test relative L2 loss between reconstructions from Zebra's VQVAE and the ground truths.

	Advection	*Heat*	*Burgers*	*Wave b*	*Combined*	*Wave 2D*	*Vorticity 2D*
VQVAE of `Zebra`	0.0003	0.0019	0.0016	0.0011	0.0022	0.010	0.017

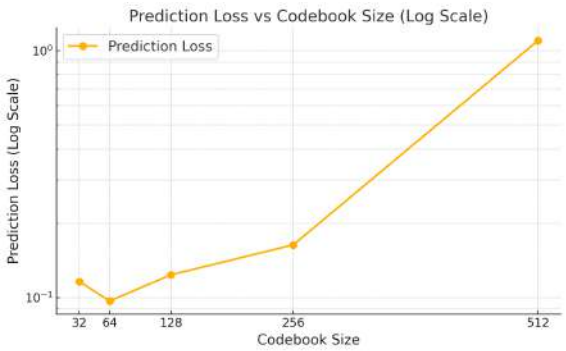


Figure C.17: One-shot accuracy vs codebook size. One-shot prediction error on the test set for various codebook sizes. Error in relative L2.

C.5 Qualitative results

We provide visualizations of the trajectories generated with `Zebra` under different settings in the following figures. **One-shot prediction**: Figure C.18, Figure C.20, Figure C.22, Figure C.24, Figure C.26, Figure C.28, Figure C.34. **Uncertainty quantification**: Figure C.19, Figure C.21, Figure C.23, Figure C.25, Figure C.23.

C.5.1 Advection

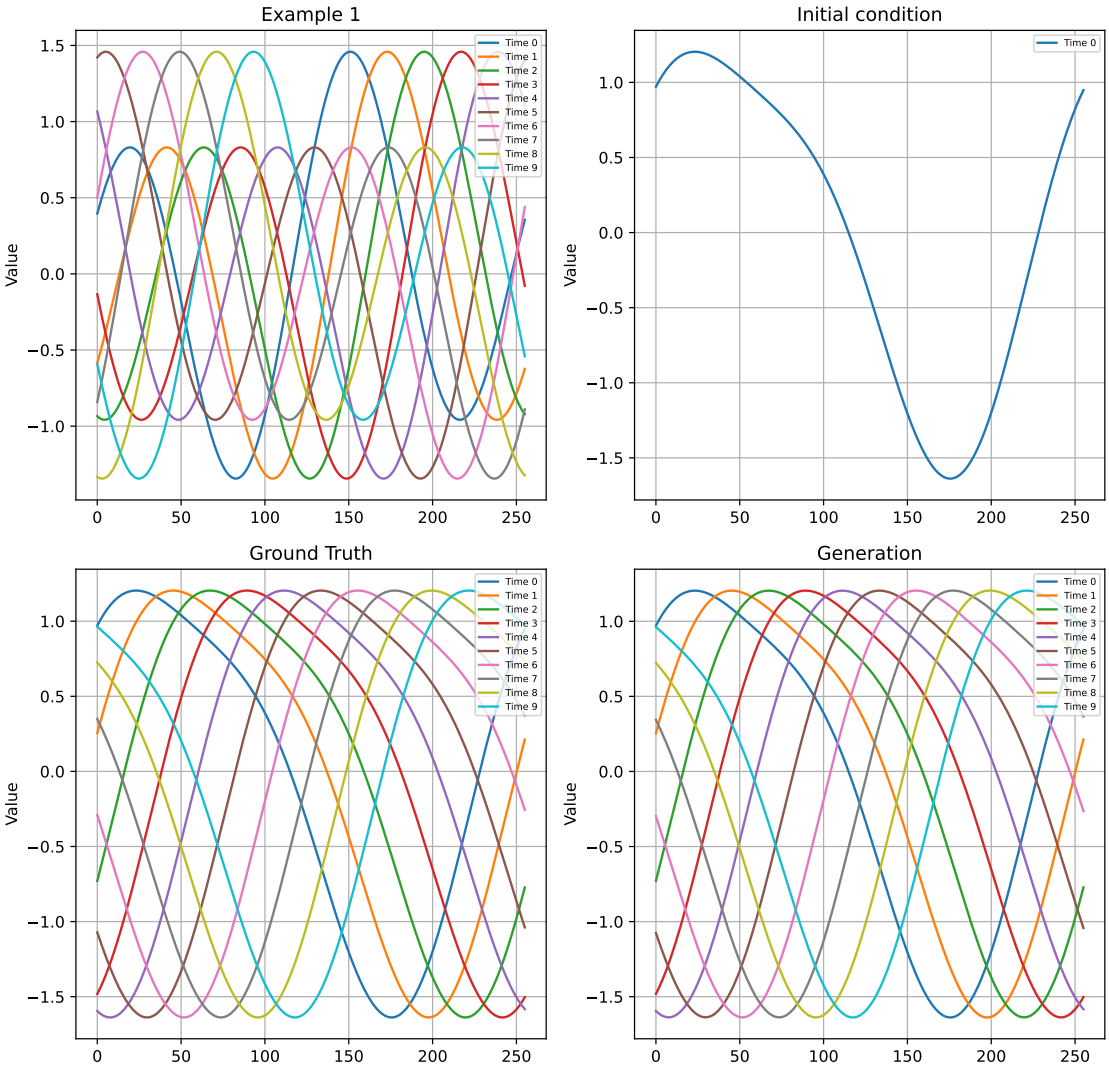


Figure C.18: One-shot adaptation on Advection

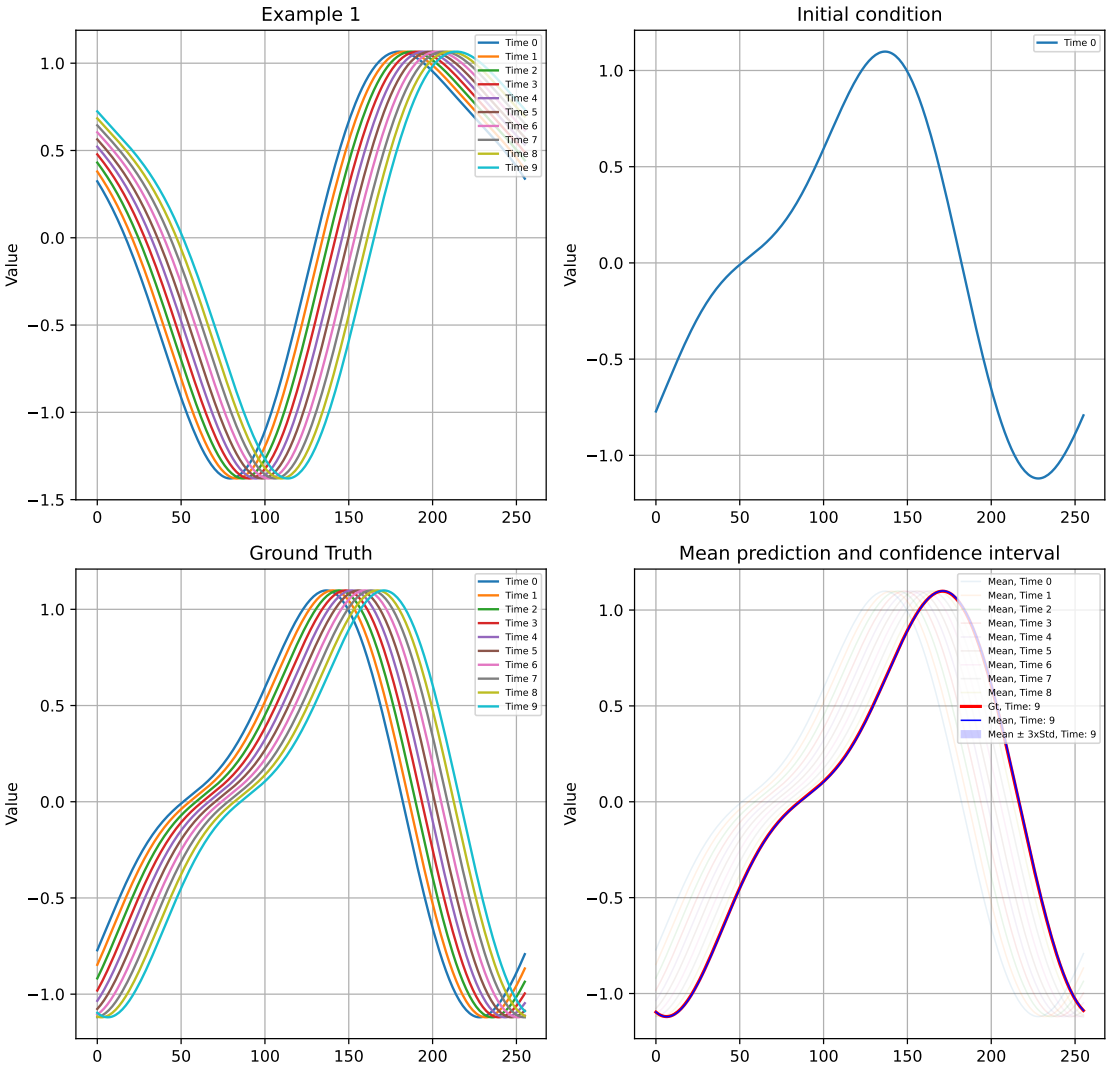


Figure C.19: Uncertainty quantification on Advection

C.5.2 Burgers

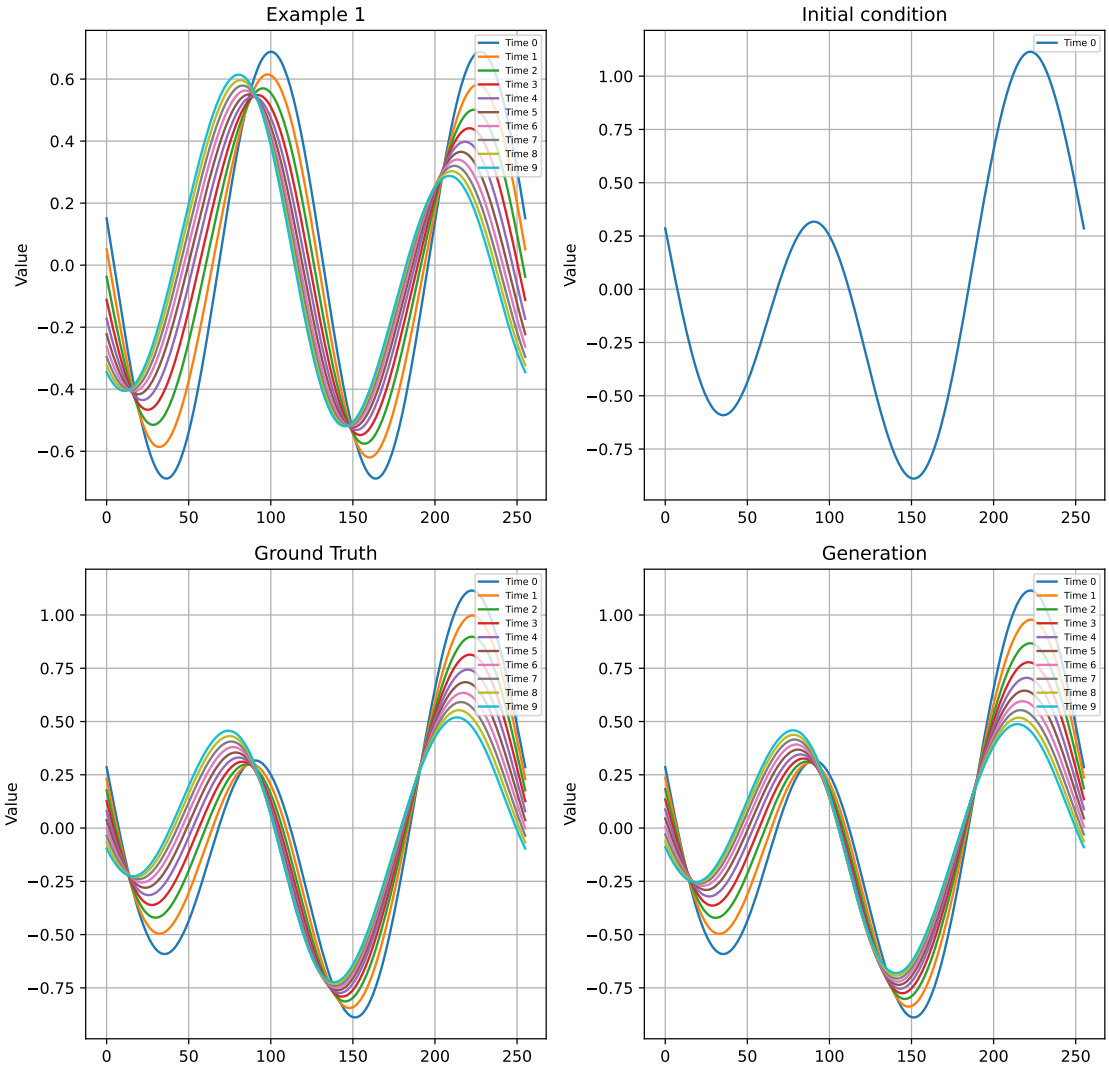


Figure C.20: One-shot adaptation on Burgers

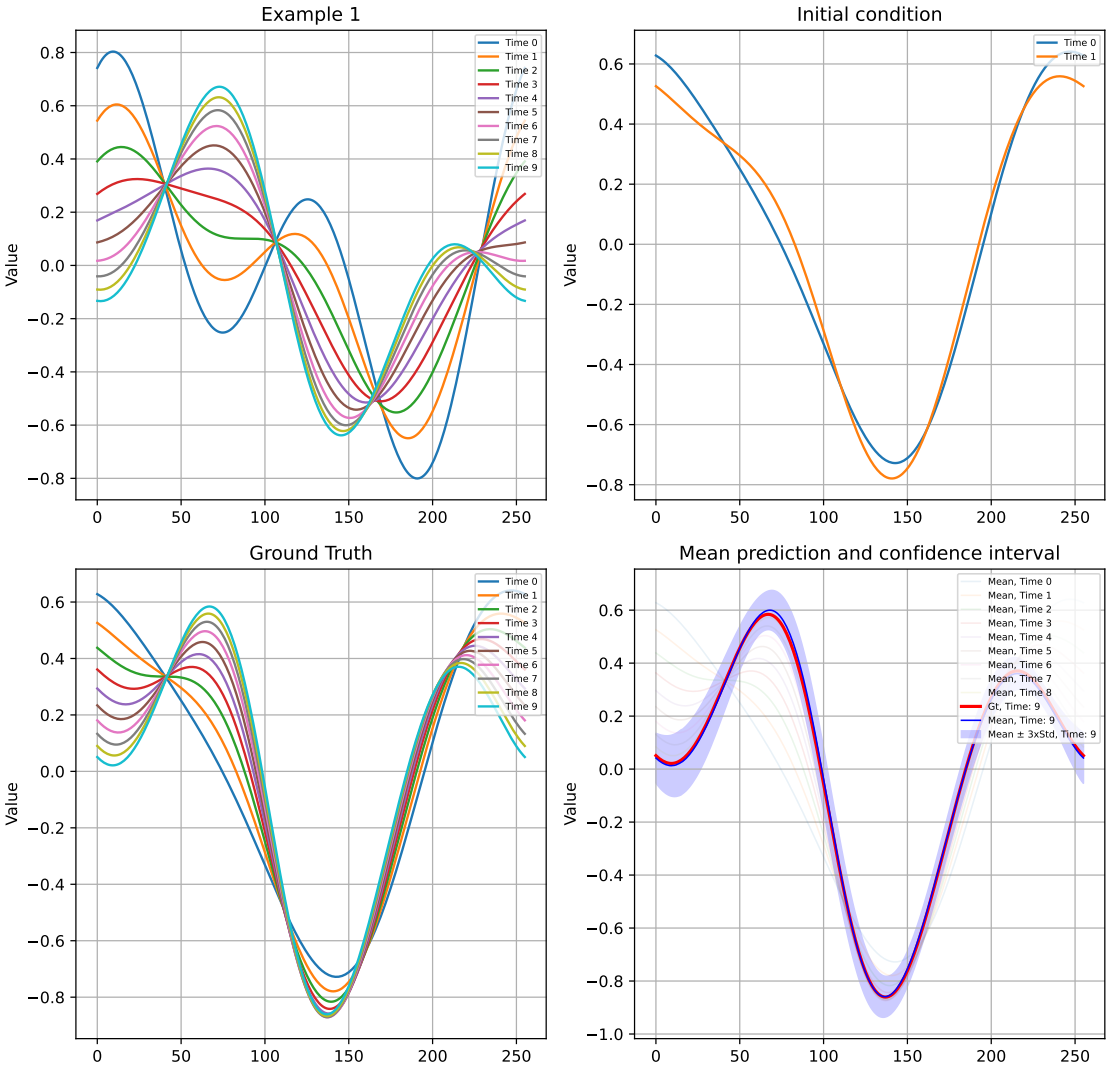


Figure C.21: Uncertainty quantification on Burgers

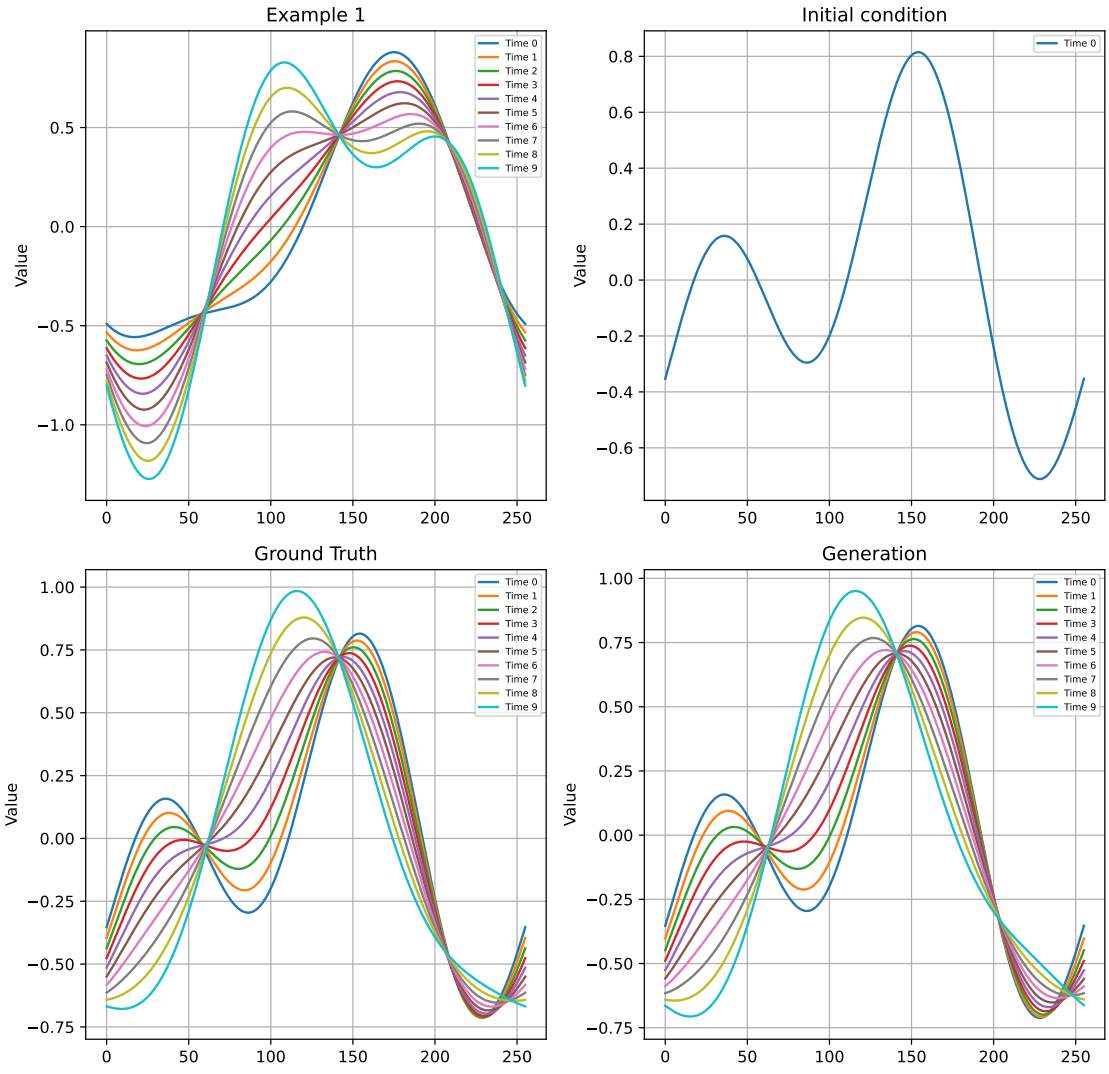


Figure C.22: One-shot adaptation on Heat

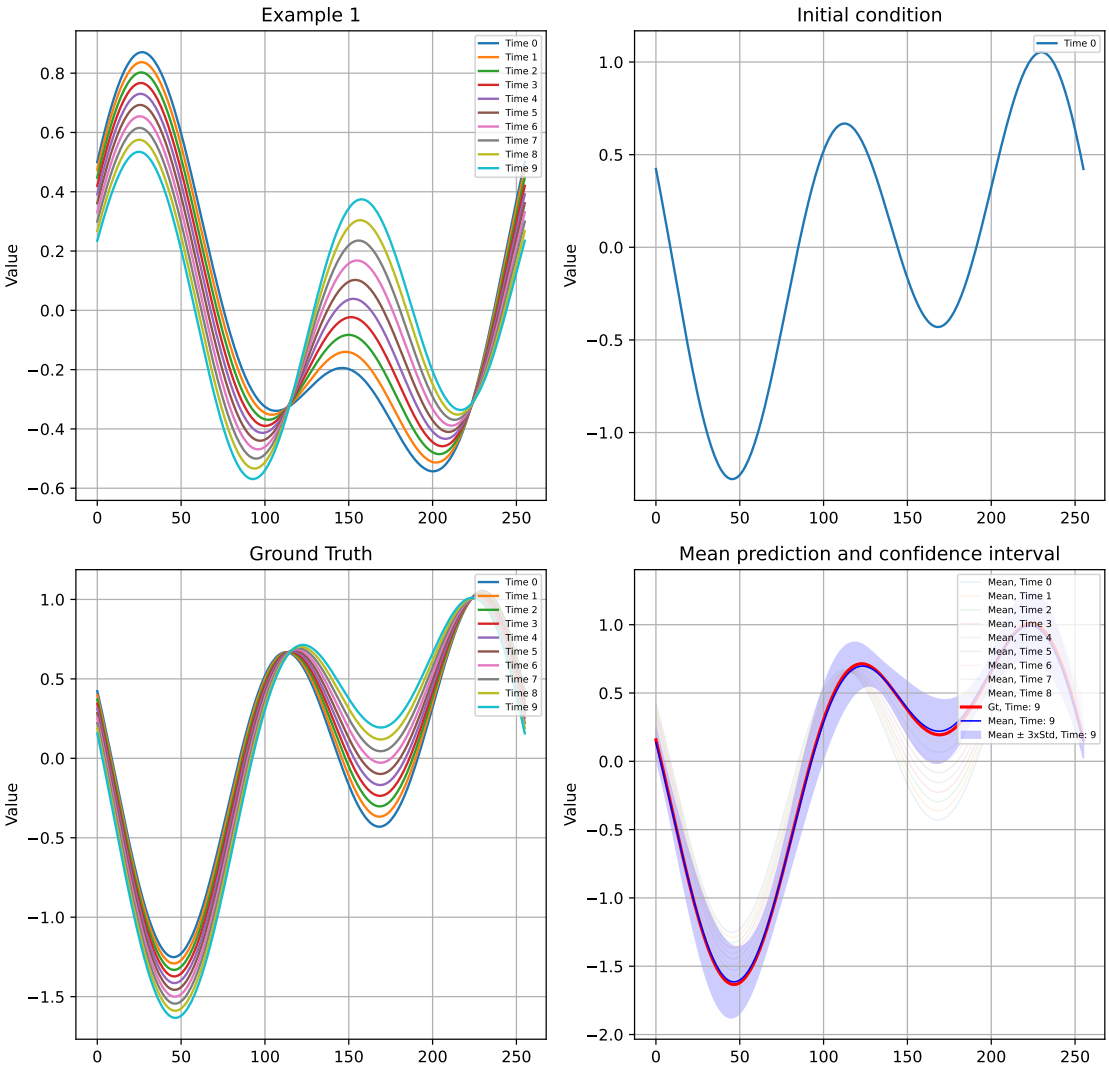


Figure C.23: Uncertainty quantification on Heat

C.5.3 Heat

C.5.4 Wave boundary

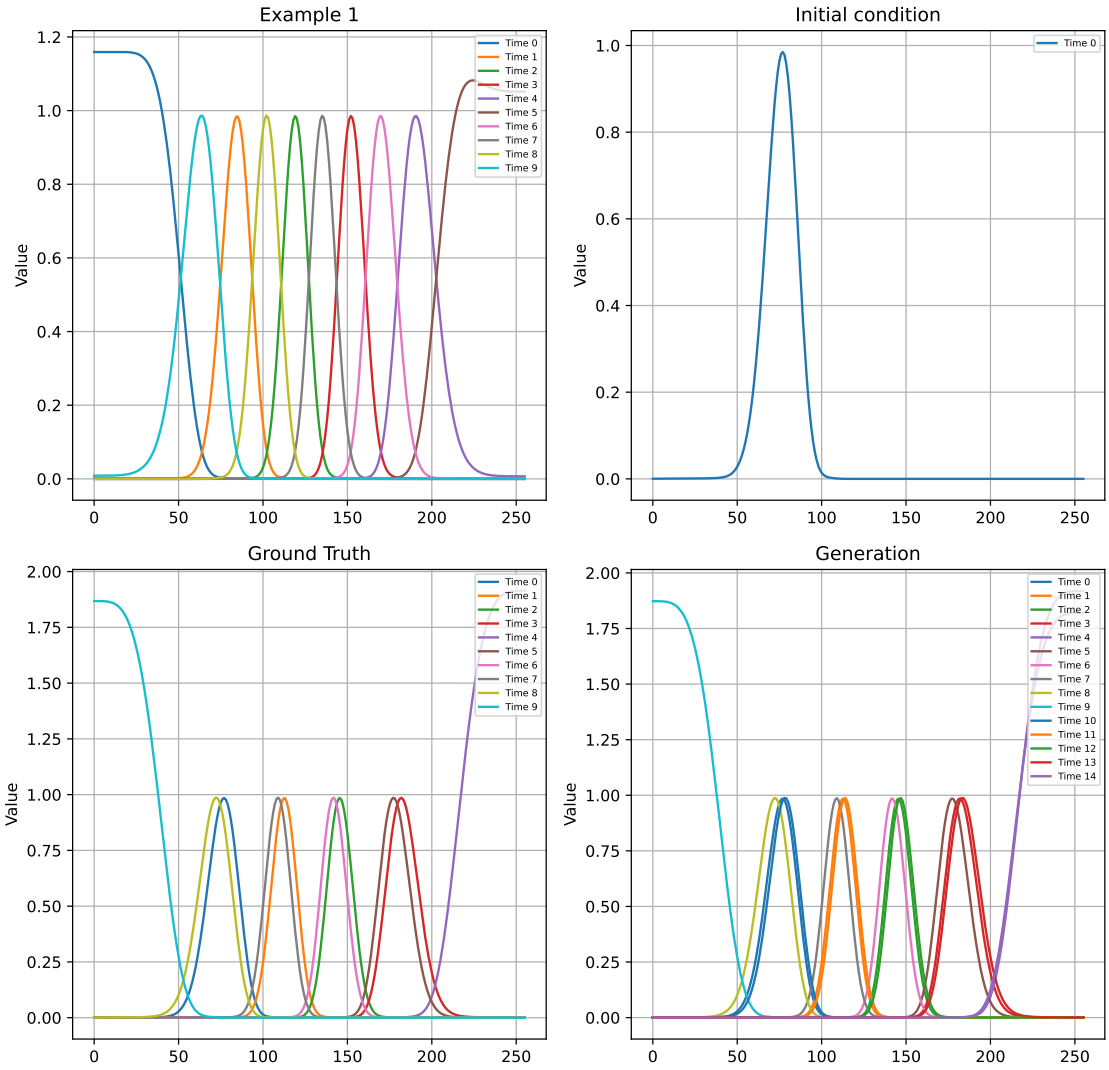


Figure C.24: One-shot adaptation on Wave b

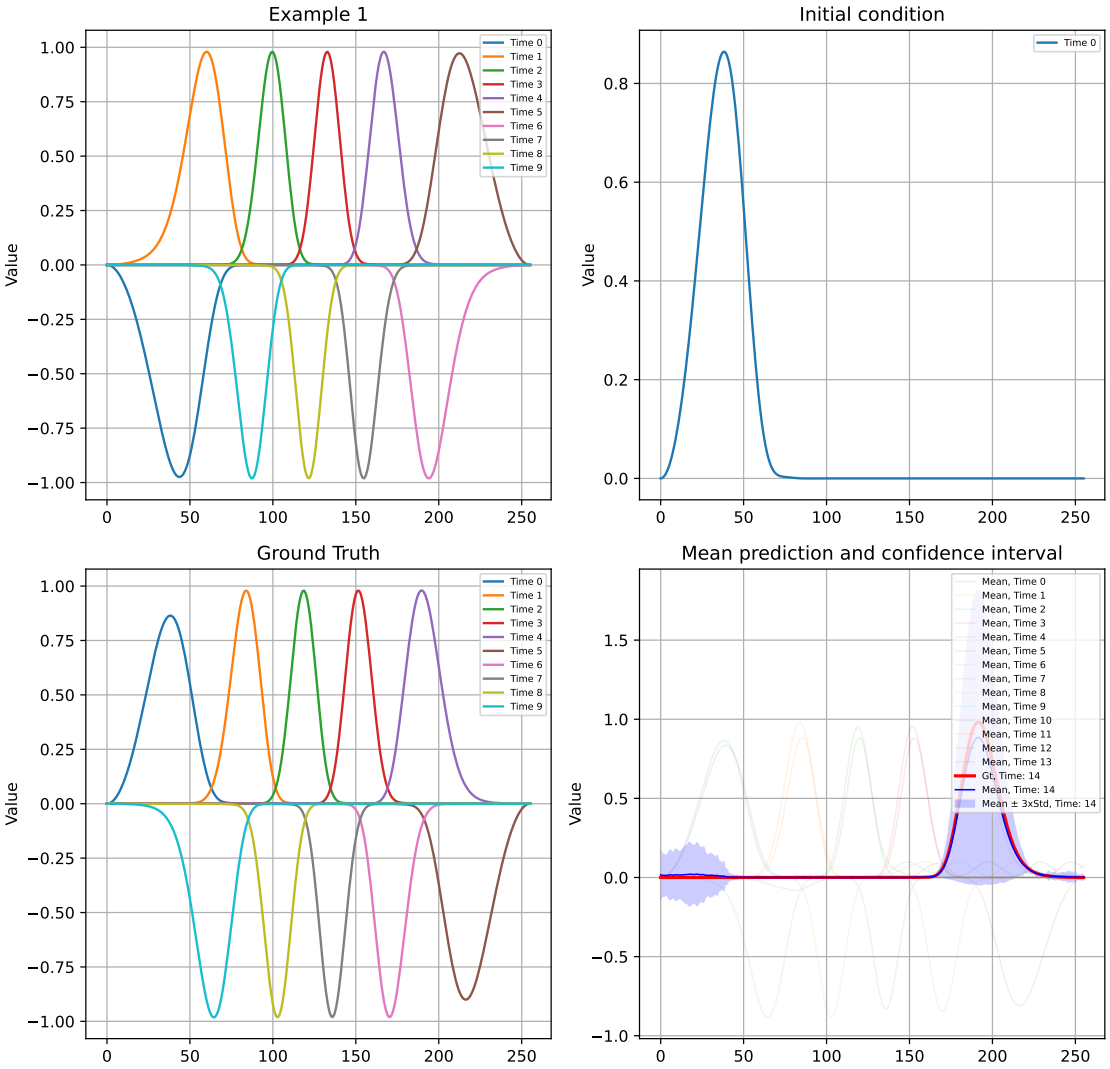


Figure C.25: Uncertainty quantification on Wave b

C.5.5 Combined equation

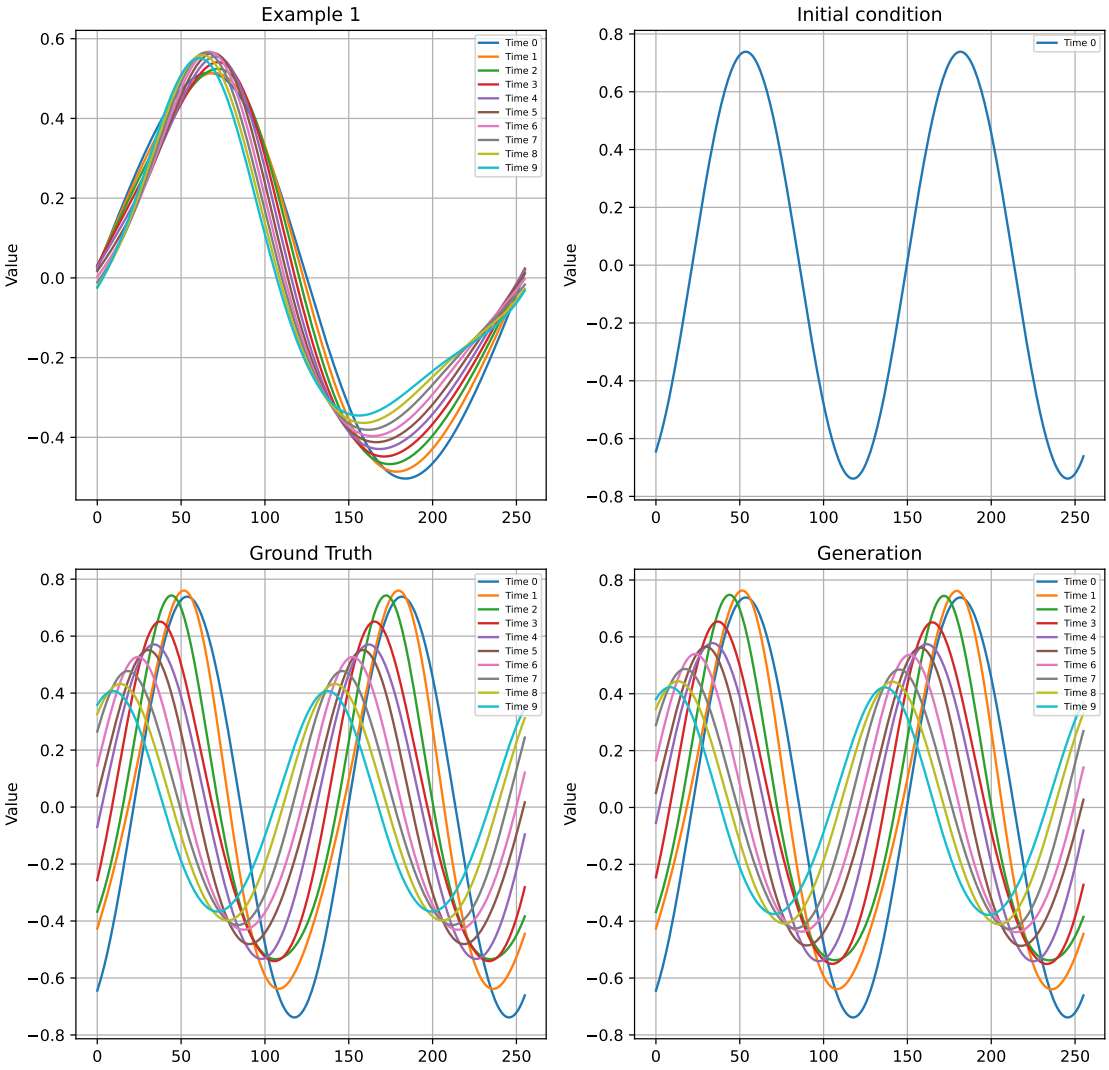


Figure C.26: One-shot adaptation on Combined

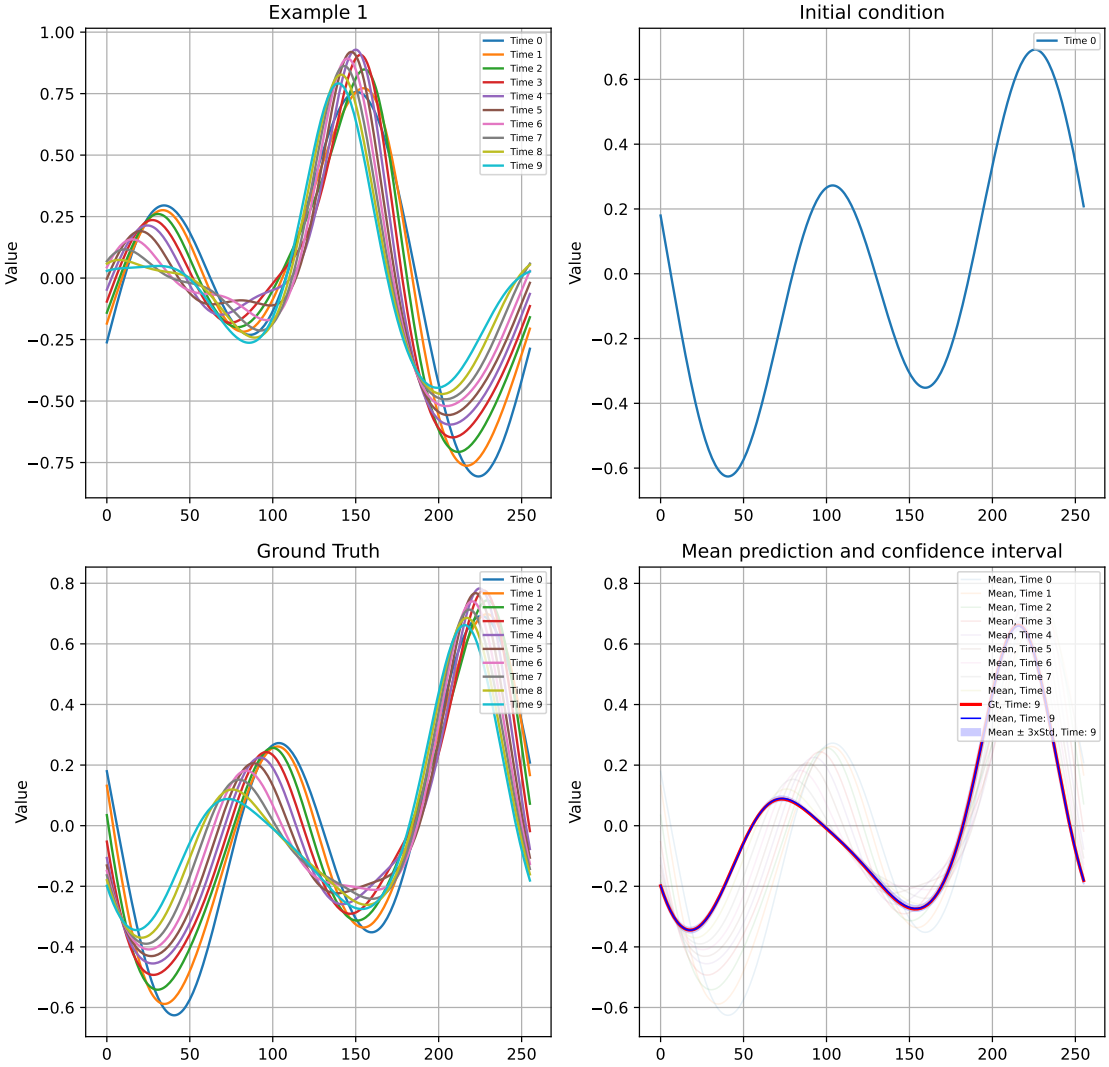


Figure C.27: Uncertainty quantification on Combined equation

C.5.6 Vorticity

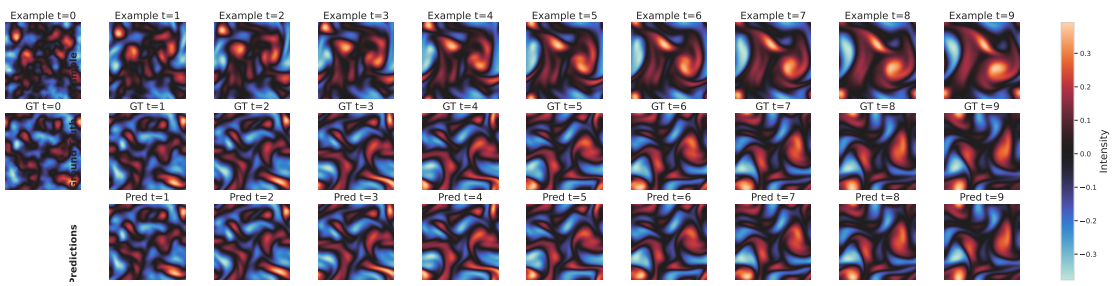


Figure C.28: One-shot adaptation on Vorticity. Example 1.

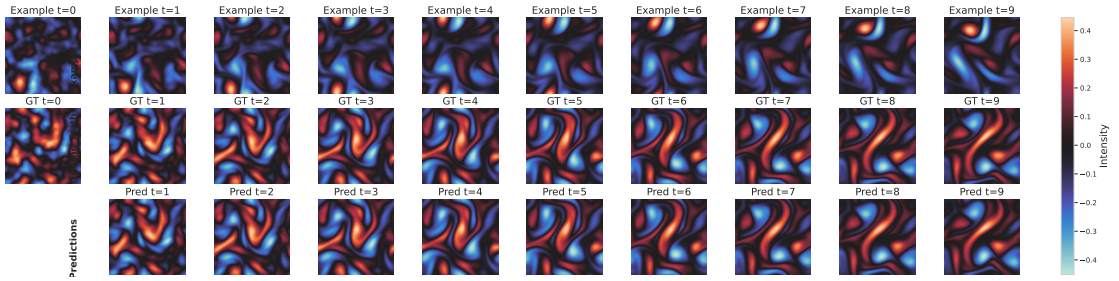


Figure C.29: One-shot adaptation on Vorticity. Example 2.

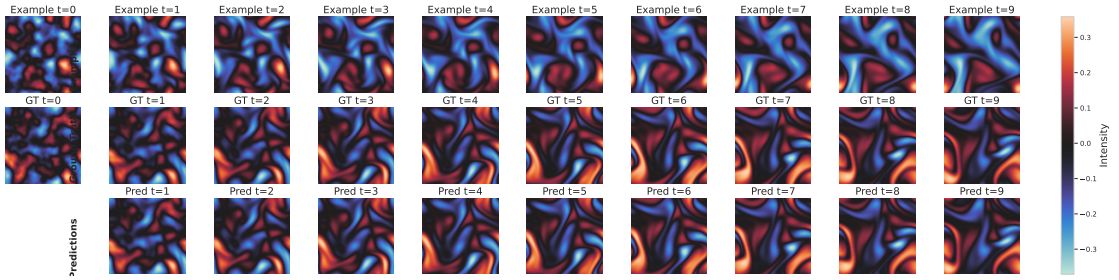


Figure C.30: One-shot adaptation on Vorticity. Example 3.

C.5.6.1 Out-of-distribution

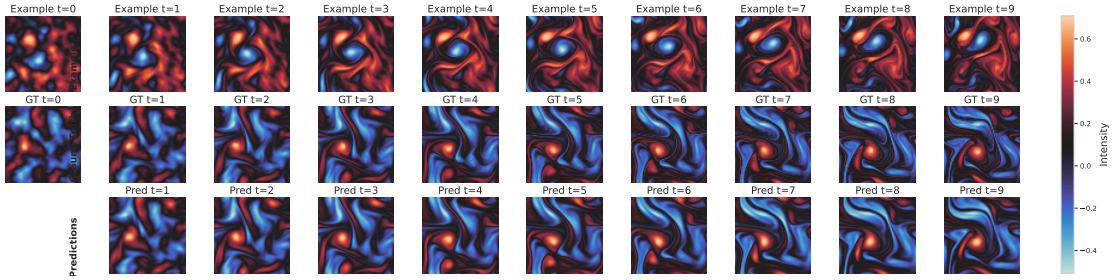


Figure C.31: One-shot OoD adaptation on Vorticity. Example 1.

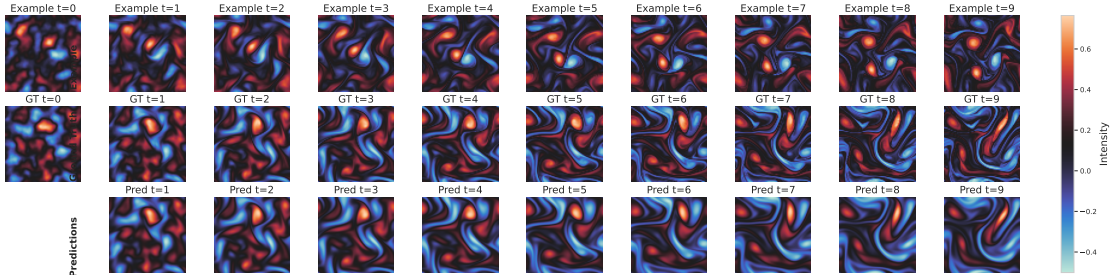


Figure C.32: One-shot OoD adaptation on Vorticity. Example 2.

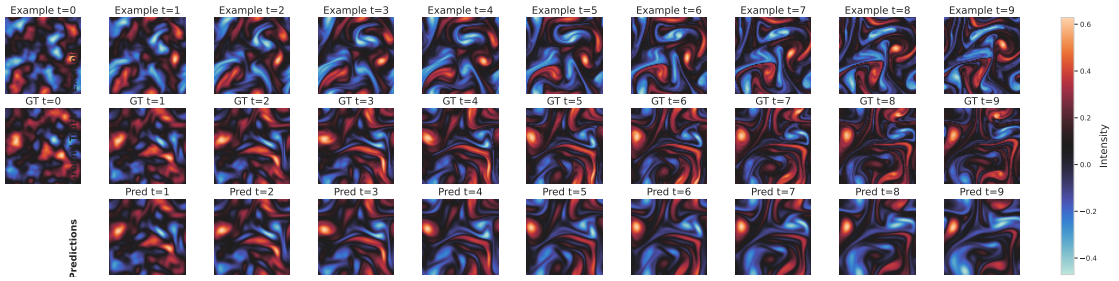


Figure C.33: One-shot OoD adaptation on Vorticity. Example 3.

C.5.7 Wave 2D

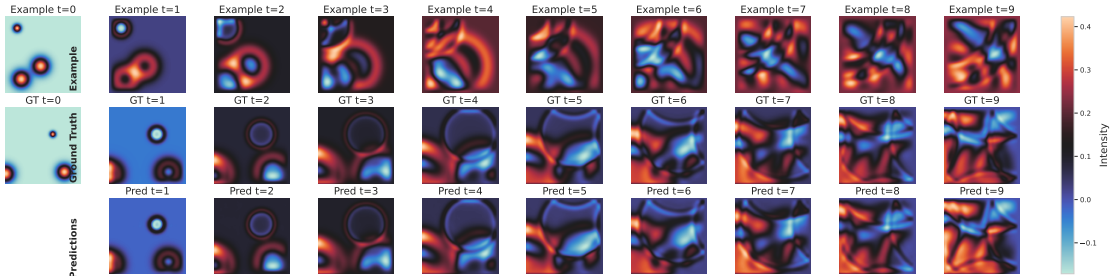


Figure C.34: One-shot adaptation on Vorticity. Example 1.

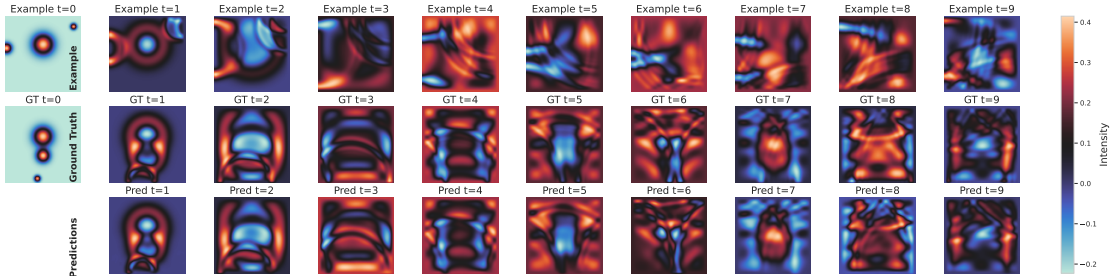


Figure C.35: One-shot adaptation on Wave2d. Example 2.

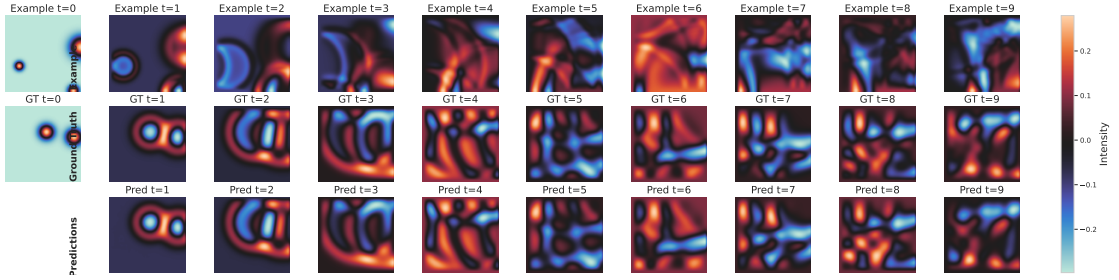


Figure C.36: One-shot adaptation on Wave2d. Example 3.