Learning Physical Operators Using Neural Operators

Vignesh Gopakumar[1,2], **Ander Gray**[2,3], **Daniel Giles**[1], **Lorenzo Zanisi**[2], **Matt J. Kusner**[4,5], **Timo Betcke**[1], **Stanislas Pamela**[2], **Marc Peter Deisenroth**[1]

[1]UCL Centre for Artificial Intelligence, [2]UK Atomic Energy Authority, [3]LIX, CNRS, École Polytechnique, [4]Polytechnique Montréal, [5]Mila - Quebec AI Institute

Abstract

Neural operators have emerged as promising surrogate models for solving partial differential equations (PDEs), but struggle to generalise beyond training distributions and are often constrained to a fixed temporal discretisation. This work introduces a physics-informed training framework that addresses these limitations by decomposing PDEs using operator splitting methods, training separate neural operators to learn individual non-linear physical operators while approximating linear operators with fixed finite-difference convolutions. This modular mixture-of-experts architecture enables generalisation to novel physical regimes by explicitly encoding the underlying operator structure. We formulate the modelling task as a neural ordinary differential equation (ODE) where these learned operators constitute the right-hand side, enabling continuous-in-time predictions through standard ODE solvers and implicitly enforcing PDE constraints. Demonstrated on incompressible and compressible Navier–Stokes equations, our approach achieves better convergence and superior performance when generalising to unseen physics. The method remains parameter-efficient, enabling temporal extrapolation beyond training horizons, and provides interpretable components whose behaviour can be verified against known physics.

Proceedings of the 29[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2026, Tangier, Morocco. PMLR: Volume 300.

1 INTRODUCTION

Partial differential equations (PDEs) serve as the mathematical foundation for modelling complex physical systems across diverse scientific and engineering applications, from fluid dynamics [Weller et al., 1998] and heat transfer [Giudicelli et al., 2024] to nuclear fusion [Hoelzl et al., 2021] and climate modelling [Danabasoglu et al., 2020]. Traditional numerical methods, such as finite element and spectral approaches [Reddy, 2006, Fornberg, 1996], while mathematically rigorous, present substantial computational challenges requiring supercomputing resources, extensive solution times and a higher carbon footprint [Keyes et al., 2013, Horwitz, 2024]. These computational limitations restrict large-scale deployment for iterative design and optimisation [Lavin et al., 2021].

Neural Operators (NO) have emerged as promising surrogate models, offering potential computational cost reductions of several orders of magnitude while maintaining considerable accuracy [Li et al., 2021, Pfaff et al., 2021, Lu et al., 2021]. However, most NOs are trained from simulation data in a supervised manner, lacking an explicit definition of the underlying PDE structure, which limits their ability to generalise outside training distributions. Their autoregressive structure further constrains temporal flexibility and extrapolation capabilities [Lee, 2023]. This work presents a novel physics-informed training framework that addresses these limitations by decomposing PDEs using operator splitting methods [Blanes et al., 2024]. We adopt a Mixture of Experts (MoE) approach [Jacobs et al., 1991] in which individual neural operators learn specific physical operators, as shown in fig. 1. The method reformulates the modelling task as a neural ordinary differential equation (ODE) with the right-hand side characterising spatial physical phenomena through a combination of neural operators and linear operators. Following operator splitting principles, our approach models non-linear operators using neural networks while approximating linear op-

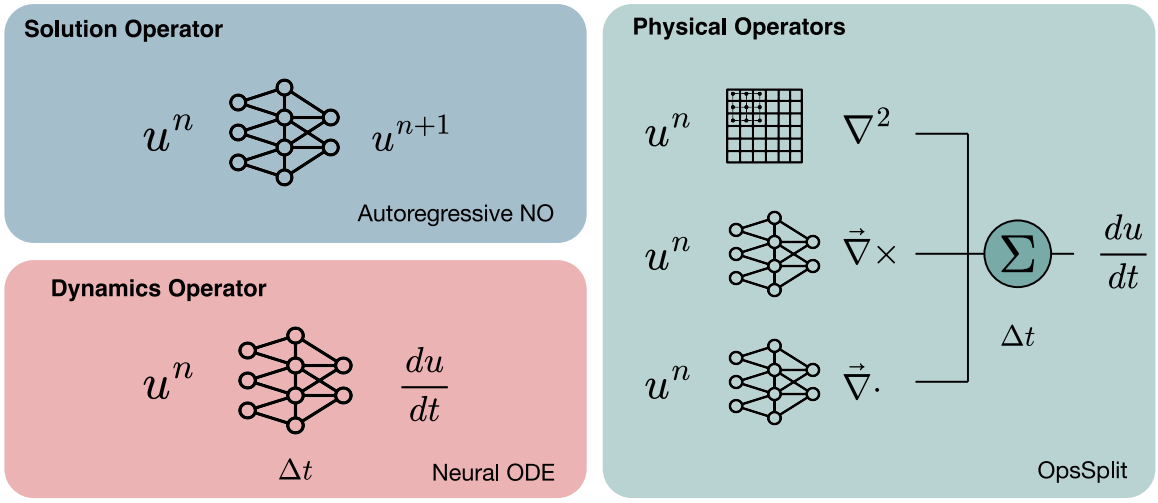


Figure 1: Comparison of operator learning approaches for PDE solving. The figure contrasts three methods: (top left) the traditional autoregressive approach where a single neural operator learns the solution operator mapping directly from state u^n to u^{n+1}; (bottom left) the neural ODE approach where a neural operator learns the dynamics operator $\frac{du}{dt}$ integrated with an ODE solver; and (right) the proposed OpsSplit method where individual neural operators learn specific physical operators ($\nabla\times, \nabla\cdot$) and are combined via operator splitting to compute $\frac{du}{dt}$ which is integrated using an ODE solver. The OpsSplit approach decomposes the PDE into its constituent physical operators, with linear operators approximated by convolutions and non-linear operators learned by neural networks, enabling physics-informed and modular PDE solving.

erators with fixed finite-difference convolutions. By learning physical operators instead of solution operators, we enforce PDE constraints during prediction and enable the construction of interpretable models whose behaviour can be numerically verified [1]. Being modular, it also allows us to add and remove operators as the physics of the system change. Our methodology provides three key advantages:

1. **Generalisable Neural Operators:** The Mixture-of-Experts structure enables neural operators to explicitly learn physical operators, creating efficient models capable of generalising beyond training regimes. The modular structure allows for the insertion and removal of operators to adapt to changing physics. By learning physical operators, we obtain better convergence for both pre-training and fine-tuning across PDEs.
2. **Continuous in Time:** A neural ODE formulation provides temporal flexibility, with the right-hand side given by combined neural and linear operators integrated using standard ODE solvers.
3. **Physics-Informed Machine Learning:** Physics consistency is inherently encoded through built-in enforcement of PDE constraints via the operator splitting structure of the prediction regime.

[1] https://github.com/gitvicky/NOs_for_POs

2 RELATED WORK

Physics-informed machine learning has created a new paradigm in computational physics by merging data with numerical models [Karniadakis et al., 2021, Lavin et al., 2021], enabling predictive forward modelling [Lippe et al., 2023], inverse modelling [Jagtap et al., 2022, Chen et al., 2021], and scientific discovery [Poels et al., 2025, Cranmer, 2023]. Neural PDE solvers have found interdisciplinary applications as surrogate models for efficient spatio-temporal PDE approximation. Physics-Informed Neural Networks (PINNs) optimise neural networks by minimising PDE residuals as loss functions [Raissi et al., 2019], while Neural Operators (NOs) learn PDE mappings by extracting dominant modes from simulation data for chosen basis functions [Lu et al., 2021]. Physics-informed Neural Operators (PINOs) combine both approaches, training NO architectures in a supervised manner before fine-tuning with PDE residual minimisation [Li et al., 2024, Rosofsky et al., 2023]. Most neural operators learn solution operators through autoregressive construction, maintaining discrete temporal evolution [Kovachki et al., 2023]. Recent works formulate continuous-time approaches using neural ODEs to learn dynamics operators [Chen et al., 2018, Serrano et al., 2023, Zhou and Barati Farimani, 2025], with extensions to differential algebraic equations where algebraic components are approximated within neural ODE frameworks [Koch

et al., 2025]. Heavily parameterised foundation models employing transformer-based architectures have enabled learning across multiple physics domains [Alkin et al., 2024, McCabe et al., 2023a, Rahman et al., 2024]. AI/ML frameworks have also led to the development of differentiable physics models, where surrogate models exist within differentiable simulation frameworks, allowing for more hybrid modelling tools [Holl et al., 2020, Um et al., 2021, Citrin et al., 2024, Bhatia et al., 2025].

3 BACKGROUND

3.1 Partial Differential Equations (PDEs)

Consider a generic formulation of a PDE modelling the spatio-temporal evolution of n field variables $u \in \mathbb{R}^n$:

$$\frac{\partial u}{\partial t} = \lambda D_X(u), \quad X \in \Omega,\ t \in [0,T], \tag{1}$$

$$u(X,t) = g, \quad X \in \partial\Omega, \tag{2}$$

$$u(X,0) = a, \quad a \in \mathcal{A}. \tag{3}$$

Here, X defines the spatial domain Ω, $[0,T]$ the temporal domain, and $\frac{\partial u}{\partial t}$ the temporal gradient. D_X represents the composite spatial derivative operator up to the PDE order. The physics coefficients are expressed via λ. The PDE has boundary condition g and initial condition a from the function space $\mathcal{A}$.

Our objective is the forward problem: solving the PDE as an initial value problem under varying coefficients, mathematically expressed as

$$\mathcal{G} : \mathcal{P} \to \mathcal{U}, \tag{4}$$

where $\mathcal{P}$ represents all characterisations of initial conditions a and PDE parameters λ, and $\mathcal{U}$ is the space of all PDE solutions over the domain.

3.2 Autoregressive Neural Operators

Neural operators (NOs) learn operator mappings across function spaces, enabling learning in infinite dimensions [Kovachki et al., 2023, Bartolucci et al., 2023]. Being discretisation-agnostic, they have found significant applications in mapping PDE initial conditions $a \in \mathcal{A}$ to solutions $u \in \mathcal{U}$ [Li et al., 2021]. A neural operator parameterised by θ learns the solution operator

$$\mathcal{U} = \mathcal{NO}_\theta(\mathcal{A}),\ u^t = \mathcal{NO}_\theta(u^{t-dt}, u^{t-2dt}, \ldots, a), \tag{5}$$

where $a = u^0$ is the initial condition. NOs couple point-wise estimations W with kernel integration κ

$$u^{n+1} = \sigma\left(W u^n(x) + \int \kappa_\phi(x,y)\, u^n(y) dy\right), \tag{6}$$

combining local linear operator W and non-local integral kernel operator K. Discretisation-agnosticism arises from kernel integration operating in continuous function spaces. Parameterising the integral operator via basis functions (Fourier modes, wavelets, Laplace eigenfunctions) yields variants like Fourier NO, Laplace NO, or wavelet NO [Li et al., 2021, Cao et al., 2023, Tripura and Chakraborty, 2023], each providing complete bases for continuous domains.

Trained autoregressively, NOs rollout field evolution with fixed temporal discretisation (dt), learning the mapping in eq. (5) as a discrete-time Markov process [Kallenberg, 1997]. Long-term PDE evolution remains challenging, with significant divergence from ground truth due to cumulative rollout error [Gopakumar et al., 2024, McCabe et al., 2023b, Koehler et al., 2024, Carey et al., 2025].

3.3 Neural ODEs

Neural ordinary differential equations (Neural ODEs) are neural networks that learn the dynamic evolution of a system's state [Chen et al., 2018]. Rather than learning the system state directly, a neural ODE approximates temporal dynamics, the rate of change. Expressed as a neural network f parameterised by θ, it learns the dynamics operator and evaluates the system state by integrating the initial value problem

$$\frac{\mathrm{d}u}{\mathrm{d}t} = f_\theta(u,t), \qquad u(T) = u(0) + \int_0^T f_\theta(u(t),t)dt. \tag{7}$$

Neural ODEs enable continuous-time dynamical modelling [Kidger, 2022] and serve as universal differential equation solvers for interdependent ODEs [Rackauckas et al., 2021]. Recently, neural operators framed as neural ODEs have achieved improved accuracy and stability for PDEs compared to autoregressive approaches [Zhou and Barati Farimani, 2025]. Any differentiable architecture satisfying the universal approximation theorem can be deployed as a neural ODE. Extensions include augmented neural ODEs for better expressivity [Dupont et al., 2019], neural SDEs [Kidger et al., 2021], and continuous graph representations [Poli et al., 2021]. While enabling continuous function modelling, neural ODEs incur additional computational costs from ODE solving.

3.4 Operator Splitting

Operator splitting methods decompose complex PDEs into simpler sub-problems solvable sequentially or in parallel [Holden et al., 2010]. The composite operator D_X in eq. (1) is split into multiple operators

representing different physical processes (e.g., convection, diffusion) that dominate field evolution within a bounded domain [Hörmander, 1983]. The decomposition pairs terms with specialised numerical techniques—advection operators with characteristic methods, diffusion operators with implicit schemes—while maintaining computational efficiency and solution accuracy. This requires domain knowledge of the physics and numerical method coupling. There may exist several ways of performing operator-splitting for a PDE requiring extensive experimentation.

Consider decomposing the spatial operator into j linear and k non-linear components

$$D_X(u) = D_{l_1}(u) + D_{l_2}(u) + ... + D_{l_j}(u) + D_{nl_1}(u) + D_{nl_2}(u) + ... + D_{nl_k}(u), \quad (8)$$

where each D_i represents a distinct component. This rewrites the PDE from eq. (1) as

$$\frac{\partial u}{\partial t} = \left(\sum_{i=1}^{j} \lambda_{l_i} D_{l_i}(u^n) + \sum_{i=1}^{k} \lambda_{nl_i} D_{nl_i}(u^n) \right), \quad (9)$$

where λ_{l_i} and λ_{nl_i} are coefficients for linear and non-linear components, respectively. Each sub-problem is solved independently over small time steps Δt using different numerical methods optimised for each operator's characteristics. Methods range from first-order (Godunov) to higher-order (Strang) splitting [MacNamara and Strang, 2016]. Decomposing linear and non-linear components enables asynchronous resource allocation, dedicating more resources to complex non-linear operators.

4 METHOD: OpsSplit

Numerical PDE solvers typically follow a discretise-then-optimise approach [Ghobadi et al., 2009], approximating spatial operators in discretised domains and solving the resulting ODEs via time integration. Neural network-based solvers like PINNs follow the inverse structure: optimise-then-discretise. Neural Operators arguably follow this paradigm [Furuya et al., 2024], with parameters optimised across fixed or varying discretisations during training [George et al., 2024], enabling discretisation-agnostic predictions. We propose a hybrid approach merging both paradigms.

The generic PDE in eq. (1) can be decomposed into linear and non-linear physical operators as in eq. (8), yielding eq. (9). Rather than traditional finite difference, integral transform, or polynomial approximations, we use neural operators $\mathbb{NO}$ for non-linear operators and convolutions with finite difference stencils $\mathbb{FD}$ for linear operators [Actor et al., 2020, Chen et al., 2024a,b, Gopakumar et al., 2025], converting eq. (9) to:

$$\frac{\mathrm{d}u}{\mathrm{d}t} = \lambda_{l_1}\mathbb{FD}_1(u) + \lambda_{l_2}\mathbb{FD}_2(u) + ... + \lambda_{l_j}\mathbb{FD}_j(u) + \lambda_{nl_1}\mathbb{NO}_1(u) + \lambda_{nl_2}\mathbb{NO}_2(u) + ... + \lambda_{nl_k}\mathbb{NO}_k(u). \quad (10)$$

Unlike autoregressive neural operators (section 3.2) that map function spaces without explicit PDE structure, or neural ODEs (section 3.3) that remain spatio-temporally continuous but approximate the entire D_X using a single operator [Zhou and Barati Farimani, 2025], our formulation (eq. (10), fig. 1) expresses a neural ODE as a linear combination of neural operators, each learning specific physical operators. This Mixture of Experts (MoE) approach [Eigen et al., 2014, Shazeer et al., 2017] dedicates expert neural operators to individual physical phenomena, enabling better efficiency and performance while facilitating disciplined scaling. Traditional neural operators suffer from spectral bias when a single parameterisation must learn dominant frequency modes across competing physical phenomena [Rahaman et al., 2019, Qin et al., 2024, Khodakarami et al., 2026]. Dedicating separate operators to each phenomenon provides better expressibility and flexibility—operators can be added or removed with changing physics.

By approximating non-linear spatial operators with neural operators, our method parallels spectral PDE methods [Polyanin and Manzhirov, 1998], producing an ODE in time once spatial gradients are approximated. The linear combination of neural operators approximates the PDE's RHS, enabling time integration via ODE solvers such as explicit Euler or Runge-Kutta methods [Ascher et al., 1997]. For Euler integration with j linear and k non-linear operators:

$$u^{n+1} = \left(\sum_{i=1}^{j} \lambda_i \mathbb{FD}_i(u^n) + \sum_{i=1}^{k} \lambda_i \mathbb{NO}_i(u^n) \right) \Delta t + u^n, \quad (11)$$

where the prediction formulation implicitly enforces PDE constraints. This constitutes a novel physics-informed approach to operator learning, structuring the differential PDE form into training and inference rather than into loss functions or architectures. As demonstrated in section 5, this framework generalises to unseen physics and novel PDE coefficient regimes. Operators are trained via supervised learning using relative LP-Loss [Kossaifi et al., 2024]:

$$\mathcal{L}_{\text{rel}}(\hat{u}^{n+1}, u^{n+1}) = \frac{\|\hat{u}^{n+1} - u^{n+1}\|_p}{\|u^{n+1}\|_p + \epsilon}, \quad (12)$$

where $\hat{u}^{n+1}, u^{n+1}$ denote prediction and ground truth at the $(n+1)$-th time instance.

4.1 Considerations for neural operator splitting

In neural settings, operator splitting requires balancing physical fidelity with computational cost. Assigning distinct Neural Operators (NOs) to each physical process substantially increases parameterisation and memory overhead. To minimise redundancy, the strategy should target vector operations invariant across the PDE family. We explore these trade-offs further with ablation studies across OpsSplit methods in appendix D. Furthermore, leveraging the fine-tuning capabilities demonstrated in appendix I, the design should prioritise modular, reusable operators that transfer across related physical systems.

5 EXPERIMENTS

We evaluate OpsSplit (learning physical operators) against Autoregressive (solution operator) and Neural-ODE (dynamics operator) approaches on incompressible and compressible Navier–Stokes equations. Performance is assessed via Normalised Root Mean Square Error (NRMSE) across four scenarios: (a) test data, (b) temporal extrapolation under identical physics, (c) out-of-distribution (OOD) with novel initial conditions and PDE coefficients, and (d) combined temporal extrapolation with OOD. NRMSE is:

$$\text{NRMSE} = \sqrt{\frac{\frac{1}{n}\sum_{i=1}^{n}(u_i - \hat{u}_i)^2}{\frac{1}{n}\sum_{i=1}^{n} u_i^2 + \epsilon}}, \tag{13}$$

where n is the number of data points, $\epsilon = 1e-6$ prevents division by zero, and $\hat{u}_i, u_i$ denote predictions and targets.

We test multiple architectures—Fourier Neural Operator (FNO) [Li et al., 2021], U-Net [Ronneberger et al., 2015, Gupta and Brandstetter, 2023], Vision Transformer (ViT) [Dosovitskiy et al., 2021, Herde et al., 2024], U-shaped Neural Operator (UNO) [Rahman et al., 2023], and Convolutional Neural Operator (CNO) [Bartolucci et al., 2023]—within each deployment method. While architecture choice impacts performance, our architecture-agnostic study focuses on deployment strategy comparisons. Performance should be compared across methods (autoregressive, neural-ODE, OpsSplit) rather than architectures. Given cost-accuracy tradeoffs [de Hoop et al., 2022], ablation studies in the appendices use FNOs exclusively.

Training: All models train for 250 epochs using Adam optimiser [Kingma and Ba, 2015] with initial learning rate 0.001, decaying by half every 50 steps. Models use LP-Loss [Gopakumar et al., 2024] on single Nvidia H100 GPUs with matched parameter counts (within each method) for fair comparison. Each experiment uses 100 PDE simulations with varied initial conditions (data efficiency ablations are demonstrated in appendix G). Models predict u^{t+1} from u^t but train by sequentially predicting 5 steps (u^{t+5}) before backpropagation. Balancing temporal learning capability against memory/compute costs, we fix the rollout length at 5 [Koehler et al., 2024], enabling consistent comparison (see appendix E for ablations). Linear range normalisation maps field values to [-1, 1]; PDE coefficients in OpsSplit use matching normalisation for physical coherence.

5.1 Incompressible Navier–Stokes

The incompressible Navier–Stokes equations are fundamental to fluid dynamics applications from aerodynamics and weather prediction to blood flow and ocean currents [Kwak and Kiris, 2009, Bauer et al., 2015, Zhang et al., 2025]. These equations pose significant computational challenges due to non-linear convection operators and coupled pressure-velocity relationships through continuity constraints, making them ideal for evaluating neural operators' multi-scale physics capture and long-term stability. Consider the two-dimensional incompressible Navier–Stokes equations:

$$\nabla \cdot \mathbf{v} = 0, \tag{14}$$
(Continuity equation)

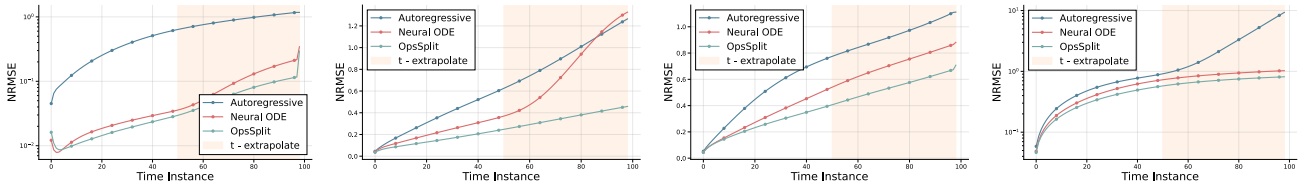
$$\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v} = \nu \nabla^2 \mathbf{v} - \nabla P, \tag{15}$$
(Momentum equation)

where $\mathbf{v} = [u, v]$ is the velocity vector of an incompressible fluid with kinematic viscosity ν under periodic boundary conditions. The system is dominated by the non-linear convection operator $(\mathbf{v} \cdot \nabla)$ accounting for momentum transport, and the linear diffusion operator ∇^2 (Laplacian) characterising viscous momentum diffusion. Pressure derives from velocity via the pressure Poisson formulation (appendix J).

OpsSplit decomposes eq. (15) into neural and linear components: $\mathbb{NO}_{conv}$ characterises combined convection and pressure Poisson effects, while $\mathbb{FD}_{\nabla^2}$ approximates diffusion via linear convolution with finite difference stencil kernels [Gopakumar et al., 2025]. Linear operators leverage well-established finite difference approximations as fixed convolutional kernels, exploiting natural correspondences between discrete differential stencils and spatial convolutions [Actor et al., 2020, Chen et al., 2024a,b], reserving neural capacity for non-linear physics while hard-coding known structures parameter-efficiently. An ablation study across various approximation methods for the linear operators are given in appendix D.1.

Operator Learning	Model	Time Stepping	OpsSplit	Parameters	Train Time (hrs:mins)	NRMSE Test	NRMSE t-extrapolate	NRMSE OOD	NRMSE OOD+t-extrapolate
Solution	FNO	Autoregressive	False	13437314	1:03	0.2770 ± 0.0491	0.6860 ± 0.26476	0.4821 ± 0.0639	0.8986 ± 0.2727
Dynamics	FNO	Euler	False	13437314	1:04	0.0325 ± 0.0165	**0.0536 ± 0.0246**	0.1276 ± 0.0081	0.3038 ± 0.0119
Physical	FNO	Euler	True	13437314	1:06	**0.0297 ± 0.0017**	0.0630 ± 0.0111	**0.1160 ± 0.0162**	**0.2185 ± 0.0846**
Solution	U-Net	Autoregressive	False	31384322	1:22	0.9508 ± 0.1369	1.3183 ± 0.1318	0.8192 ± 0.2007	1.1872 ± 0.2534
Dynamics	U-Net	Euler	False	31384322	1:22	0.1006 ± 0.0133	0.2212 ± 0.0315	0.2823 ± 0.0154	0.5135 ± 0.0331
Physical	U-Net	Euler	True	31384322	1:26	**0.0887 ± 0.0116**	**0.1996 ± 0.0190**	**0.2705 ± 0.0060**	**0.4550 ± 0.0181**
Solution	ViT	Autoregressive	False	11405510	3:42	0.2335 ± 0.0468	0.4234 ± 0.0468	0.7505 ± 0.0713	0.8996 ± 0.0647
Dynamics	ViT	Euler	False	11405510	3:43	0.1034 ± 0.0081	0.2179 ± 0.0191	0.3747 ± 0.0199	0.5984 ± 0.0699
Physical	ViT	Euler	True	11405510	3:45	**0.0157 ± 0.0008**	**0.0822 ± 0.0016**	**0.3065 ± 0.0125**	**0.4742 ± 0.0195**
Solution	CNO	Autoregressive	False	1420458	9:43	0.5094 ± 0.0611	0.9002 ± 0.1620	0.5831 ± 0.0875	3.1951 ± 0.3515
Dynamics	CNO	Euler	False	1420458	9:45	0.2985 ± 0.0478	0.7122 ± 0.0997	0.4387 ± 0.0834	0.9055 ± 0.1177
Physical	CNO	Euler	True	1420458	9:52	**0.2797 ± 0.0475**	**0.5200 ± 0.0624**	**0.3634 ± 0.0545**	**0.6917 ± 0.1245**
Solution	UNO	Autoregressive	False	1397442	2:23	0.0912 ± 0.0100	0.2856 ± 0.0543	0.6190 ± 0.0867	0.8215 ± 0.1314
Dynamics	UNO	Euler	False	1397442	2:23	0.0141 ± 0.0018	**0.1747 ± 0.0297**	**0.2986 ± 0.0358**	**0.6221 ± 0.1120**
Physical	UNO	Euler	True	1397442	2:24	**0.0130 ± 0.0020**	0.1857 ± 0.0204	0.3231 ± 0.0614	0.7076 ± 0.0991

Table 1: Incompressible Navier–Stokes: Performance comparison across methods and model architectures. The models and methods are set up to have comparable parameter sizes while exploring a certain architecture. Best performance within each test setting is given in bold. Our method of using neural operators to learn physical operators offers the best performance across most architectures. OOD refers to a different parameterisation of the initial condition and viscosity than that used for training (appendix J).



(a) FNO: in-distribution (b) FNO: out-of-distribution (c) CNO: in-distribution (d) CNO: out-of-distribution

Figure 2: Rollout error for incompressible Navier–Stokes equations. FNO (figs. 2a and 2b) and CNO (figs. 2c and 2d) predictive error growth for in-distribution and out-of-distribution cases. The temporal extrapolation region is shaded orange. OpsSplit accumulates fewer errors and provides more stable temporal rollout than autoregressive and neural ODE methods across architectures and scenarios.

$$\frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = -\mathbb{NO}_{conv}(\mathbf{v}) + \nu\mathbb{FD}_{\nabla^2}(\mathbf{v}) \tag{16}$$

Equation (16) reformulates momentum as a neural ODE explicitly separating non-linear (convection via $\mathbb{NO}_{conv}$, including pressure Poisson effects from eq. (35)) and linear physics (diffusion via fixed stencil $\mathbb{FD}_{\nabla^2}$). This converts the PDE to ODEs integrable via standard solvers. During training, velocity fields pass through both operators; their weighted combination (scaled by ν) provides $\frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t}$, integrated forward for predictions. Training data generated by solving eqs. (14) and (15) on $x \in [0,1], y \in [0,1], t \in [0,0.5]$ via spectral solver [Canuto et al., 2007]. Temporal extrapolation tests use data evolved to $t = 1.0$; OOD tests use different initial condition parameterisations and viscosity conditions. See appendix J for physics, solver, parameterisation, and model details.

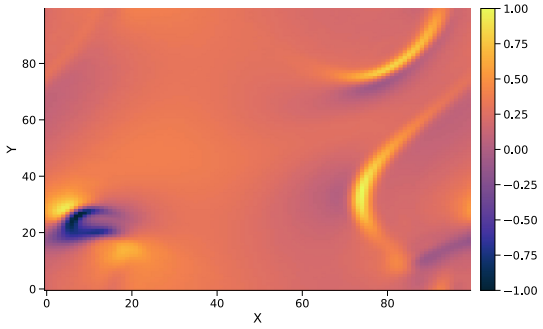
Table 1 quantitatively compares OpsSplit (physical) against autoregressive (solution) and neural ODE (dynamics) methods. OpsSplit demonstrates superior performance across architectures, particularly in OOD scenarios with novel physics and initial conditions. Figure 2 shows FNO and CNO rollout errors across all deployment methods—OpsSplit exhibits the lowest error growth in both in-distribution and OOD scenarios. Figure 3 compares the learned neural convection operator (fig. 3a) with that utilised in the numerical solver (fig. 3b). OpsSplit's NO explicitly learns general behaviour of the physical operator, enabling physics-informed, interpretable models (see appendix C for further studies). Figure 4 evaluates physical consistency of each method via physics residual error of the continuity equation (eq. (14)). Surprisingly, neural ODE struggles with the physics as we extrapolate temporally, while autoregressive and OpsSplit methods strive to keep the inconsistencies to a minimum.

5.2 Compressible Navier–Stokes

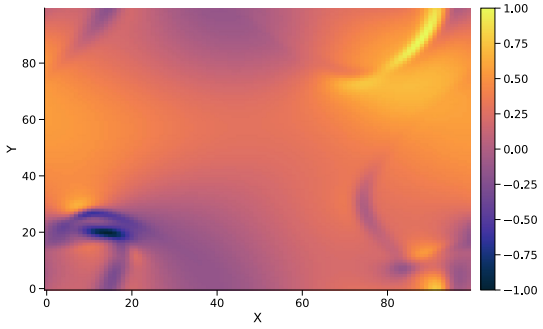
The compressible Navier–Stokes equations are essential for modelling high-speed flows in aerospace, shock waves, and astrophysical phenomena where density variations are significant [Liu, 1986, Hujeirat and Rannacher, 1998]. This system challenges neural PDE

Operator Learning	Model	Time Stepping	OpsSplit	Parameters	Train Time (hrs:mins)	NRMSE			
						Test	t-extrapolate	OOD	OOD+t-extrapolate
Solution	FNO	Autoregressive	False	13437828	1:53	0.0938 ± 0.0131	0.2462 ± 0.0443	0.1209 ± 0.0133	0.2792 ± 0.0530
Dynamics	FNO	Euler	False	13437828	1:56	**0.0245 ± 0.0039**	**0.0854 ± 0.0102**	0.0763 ± 0.0130	0.1185 ± 0.0154
Physical	FNO	Euler	True	13454787	3:08	0.0602 ± 0.0090	0.0959 ± 0.0182	**0.0573 ± 0.0063**	**0.0751 ± 0.0105**
Solution	U-Net	Autoregressive	False	31385604	1:53	0.5963 ± 0.0716	1.1914 ± 0.2025	0.6069 ± 0.1153	1.2384 ± 0.1610
Dynamics	U-Net	Euler	False	31385604	1:54	0.1222 ± 0.0196	0.2912 ± 0.0320	0.1477 ± 0.0222	0.2989 ± 0.0538
Physical	U-Net	Euler	True	31384322	2:47	**0.0781 ± 0.0109**	**0.1081 ± 0.0184**	**0.0799 ± 0.0096**	**0.1149 ± 0.0218**
Solution	ViT	Autoregressive	False	11642836	7:54	0.2337 ± 0.0351	0.4323 ± 0.0476	0.2852 ± 0.0513	0.5109 ± 0.0664
Dynamics	ViT	Euler	False	11642836	7:55	**0.0557 ± 0.0089**	0.2761 ± 0.0387	0.0963 ± 0.0183	0.3044 ± 0.0365
Physical	ViT	Euler	True	11609798	10:13	0.0796 ± 0.0135	**0.1296 ± 0.0143**	**0.0818 ± 0.0123**	**0.1396 ± 0.0251**
Solution	UNO	Autoregressive	False	1398468	3:51	0.2689 ± 0.0323	0.7733 ± 0.1315	0.3028 ± 0.0454	0.8366 ± 0.1590
Dynamics	UNO	Euler	False	1398468	3:51	**0.0142 ± 0.0020**	**0.0660 ± 0.0073**	0.0706 ± 0.0127	0.0831 ± 0.0108
Physical	UNO	Euler	True	2400563	11:09	0.0563 ± 0.0090	0.0861 ± 0.0164	**0.0574 ± 0.0069**	**0.0752 ± 0.0113**

Table 2: Compressible Navier–Stokes: Performance comparison across methods and model architectures. The models and methods are set up to have comparable parameter sizes while exploring a certain architecture. Best performance within each test setting is given in bold. Our method of using neural operators to learn physical operators offers the best performance across most architectures. OOD refers to a different parameterisation of the initial condition and adiabatic index than that used for training (appendix K). The higher computational times for OpsSplit is observed due to the several forward passes required to estimate the physical operators.



(a) Convection Operator: Neural



(b) Convection Operator: Numerical

Figure 3: Neural operator learned convection (fig. 3a) versus numerical method. NO captures convection nuances and advection-driven flow trends. Qualitative interpretability study: learned convection exists in latent space; numerical convection shown in physical space, normalised to [-1, 1].

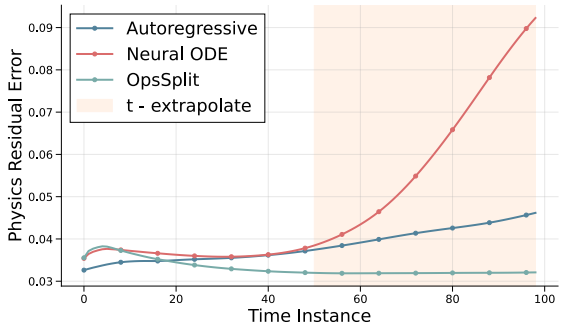


Figure 4: Continuity equation (eq. (14)) violation across FNO predictions for OOD modeling.

solvers through tight density-velocity-pressure coupling. They explore discontinuities and shock formation while preserving conservation laws across multiple interacting physical processes. Consider the Euler formulation of the compressible Navier–Stokes equations, which govern adiabatic, inviscid fluid flow:

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot (\rho \mathbf{v}), \tag{17}$$

(Mass conservation / Continuity)

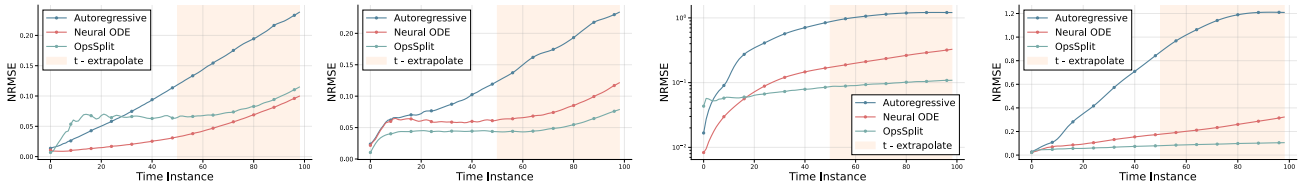
$$\frac{\partial \mathbf{v}}{\partial t} = -(\mathbf{v} \cdot \nabla)\mathbf{v} - \frac{1}{\rho}\nabla P, \tag{18}$$

(Momentum conservation)

$$\frac{\partial P}{\partial t} = -\mathbf{v} \cdot \nabla P - \gamma P(\nabla \cdot \mathbf{v}). \tag{19}$$

(Pressure evolution)

These equations model density ρ, velocity $\mathbf{v} = [u, v]$, and pressure P evolution for a gas with adiabatic index (specific heat ratio) γ under periodic boundary conditions. Density evolution applies divergence to momentum; convection and pressure gradient terms control momentum. Density-weighted pressure terms ensure greater acceleration in low-density regions. Energy (pressure) depends on pressure advection ($\mathbf{v} \cdot \nabla P$) and compression/expansion ($\gamma P \nabla \cdot \mathbf{v}$), modulating pressure via velocity divergence.



(a) FNO: in-distribution (b) FNO: out-of-distribution (c) UNet: in-distribution (d) UNet: out-of-distribution

Figure 5: Rollout error for the compressible Navier–Stokes equations. Similar to fig. 2, show the rollout error of various methods for an FNO and U-Net for both in and out-of-distribution scenarios. In most cases, across neural operator architectures, our method of deploying operator splitting to learn the physical operator accumulates less error and provides a more stable temporal rollout than autoregressive and neural ODE-based methods.

OpsSplit uses two neural operators to learn vector divergence and convection operations explicitly. The vector divergence operator $\mathbb{NO}_{\nabla\cdot}$ (eqs. (20) and (22)) learns spatial momentum and energy evolution relative to density and pressure, with adiabatic index influence explicitly applied over the divergence of pressure-velocity as shown ineq. (22). The convection operator (eq. (21)) learns fluid advection similarly to eq. (16). Pressure gradient terms use linear convolution with higher-order finite difference stencils as a fixed kernel $\mathbb{FD}_{\nabla}$. For temporal integration stability, density weighting in eq. (18) uses logarithms to avoid division by zero [Shebalin, 1993, Erlebacher et al., 1990]. See appendix K for physics, solver, parameterisation, and model details.

$$\frac{\mathrm{d}\rho}{\mathrm{d}t} = -\mathbb{NO}_{\nabla\cdot}(\rho, \mathbf{v}), \tag{20}$$

$$\frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = -\mathbb{NO}_{conv}(\mathbf{v}) - \ln(\rho)\,\mathbb{FD}_{\nabla}(P), \tag{21}$$

$$\frac{\mathrm{d}P}{\mathrm{d}t} = -\gamma\,\mathbb{NO}_{\nabla\cdot}(P, \mathbf{v}). \tag{22}$$

Table 2 demonstrates OpsSplit's superior performance across most architectures. While all methods achieve comparable in-distribution accuracy, OpsSplit outperforms autoregressive and neural ODE approaches when generalising to unseen parameter regimes and extrapolating in time. Figure 5 shows OpsSplit maintains substantially lower and more stable error growth throughout temporal evolution in OOD scenarios, stemming from explicit PDE constraint enforcement via operator splitting. Note that despite comparable parameters, OpsSplit's dual NOs increase training time versus autoregressive and neural ODE methods: 1.5× (FNO) to 3× (UNO).

5.3 Unstructured Neural Operators

We extend our OpsSplit framework to irregular geometries by replacing the nonlinear operators with graph-based neural operators. These neural operators rely on message passing across an unstructured mesh to learn the evolution of the spatial dynamics within the PDE [Li et al., 2020, 2023, Brandstetter et al., 2022]. We follow the CylinderFlow experiment from Pfaff et al. [2021], where the authors develop a graph neural network to learn the spatio-temporal evolution of vortex shedding as fluid passes over a cylinder. The experiment is governed by the incompressible Navier–Stokes equations given in eqs. (14) and (15). In implementing OpsSplit, we adopt the same splitting as in eq. (16), but employ a neural operator for both convection and diffusion (eq. (23)). By modelling the linear operators with a learnable graph-based operator, we bypass the non-trivial challenge of implementing finite differences on unstructured meshes.

$$\frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = -\mathbb{NO}_{\mathrm{conv}}(\mathbf{v}) + \nu\mathbb{NO}_{\mathrm{diff}}(\mathbf{v}) \tag{23}$$

The CylinderFlow dataset comprises 1000 simulations of fluid flow, all with identical viscosity coefficients but featuring distinct geometries and mesh structures for each simulation. Using AR, NODE, and OpsSplit, we model the spatio-temporal evolution of velocities, training up to the halfway point of each simulation and then extrapolating further during evaluation.

The results in table 3 demonstrate that OpsSplit achieves superior performance on both in-distribution and out-of-distribution test cases compared to both the neural ODE and the autoregressive baseline. This enhanced performance comes at a moderate computational cost, with training time somewhat higher due to the multiple neural operators required. These results confirm that incorporating physical structure through operator splitting translates effectively to unstructured geometries, offering substantial gains in predictive accuracy for complex fluid flow scenarios.

Operator Learning	Model	Time Stepping	OpsSplit	Parameters	Train Time (hrs:mins)	NRMSE In-Distribution	NRMSE Out-of-Distribution
Solution	GNO	Autoregressive	False	330,370	6:25	0.4131 ± 0.0475	0.5697 ± 0.0530
Dynamics	GNO	Euler	False	330,370	6:26	0.3276 ± 0.0398	0.4510 ± 0.0571
Physical	GNO	Euler	True	660,740	9:31	$\mathbf{0.2963 \pm 0.0043}$	$\mathbf{0.3621 \pm 0.0510}$

Table 3: Incompressible Navier–Stokes on irregular geometries: Performance comparison across methods and model architectures. Best performance within each test setting is shown in bold. Our method of using neural operators to learn physical operators achieves the best performance. In-distribution refers to modelling within the training regime, whereas out-of-distribution refers to extrapolation further in time on a different validation dataset. The higher computational time for OpsSplit is due to the multiple models required for each physical operator.

6 DISCUSSION

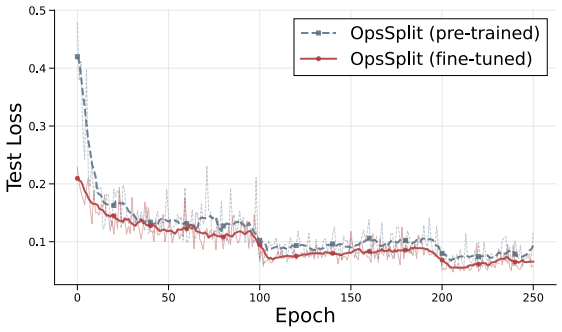


Figure 6: Loss convergence for OpsSplit models on incompressible Navier–Stokes. The **pre-trained** model learns all operators from scratch. The **fine-tuned** model initialises the convection operator with trained weights from the compressible case, showing quicker convergence. For more details, refer appendix I.

Strengths By explicitly learning physical operators, OpsSplit invokes a modular architecture where a specialised neural network approximates each operator. This enables generalisable neural operators reusable across multiple physical scenarios while facilitating efficient transfer learning as shown in fig. 6. The modular structure supports dynamic insertion and removal of operators as physics changes, and enhances model interpretability by isolating failure regions to specific physical operators. Like neural ODEs, our approach maintains explicit awareness of temporal dynamics, enabling stable long-term rollouts and continuous-in-time predictions that support both interpolation and extrapolation. Our physics-informed decomposition provides a novel constraint enforcement mechanism: since the prediction regime in eq. (11) aligns with the PDE definition, it naturally implements soft physical constraints. OpsSplit achieves parameter and computational efficiency by approximating only non-linear operators with neural networks while using linear methods for linear operators, reducing modelling complexity. The decomposition also enables direct parallelisation and efficient training procedures.

Weaknesses Physics-informed machine learning requires a complete understanding of the underlying equations and domain expertise for operator splitting. Implementation demands explicit model configuration for non-linear operators and convolution setup with finite difference stencils for linear operators, increasing development effort and introducing approximation errors. Critically, no general splitting rule exists, and different splitting of the same problem yield vastly different performances [McLachlan and Quispel, 2002] (refer appendix D). When systems possess multiple geometric properties preserved during evolution, different splittings preserve different properties, making it difficult to find one that preserves most. For complex multi-physics settings, structuring individual neural operators for each PDE operator may lead to heavily parameterised models with elevated computational and memory overheads. The ODE formulation incurs higher computational costs than autoregressive methods due to time integration. While we currently explore periodic boundary conditions, the method can be adapted to other boundary conditions by augmented padding schemes [Alguacil et al., 2021, McCabe et al., 2025].

7 CONCLUSION

We present a novel method for solving PDEs using neural operators that enforce physical constraints while maintaining continuity in space and time. By learning non-linear physical operators rather than solution operators (autoregressive) or dynamics operators (neural ODE), our approach achieves physics-informed, parameter-efficient modelling with robust out-of-distribution generalisation. The physics-informed, modular design also opens avenues for inverse modelling, system identification, and accelerating the transition from simulation to experimental data, a new paradigm in bridging the *sim2real* gap.

References

H. G. Weller, G. Tabor, H. Jasak, and C. Fureby. A tensorial approach to computational continuum mechanics using object-oriented techniques. *Comput. Phys.*, 12(6):620–631, November 1998. ISSN 0894-1866. doi: 10.1063/1.168744. URL https://doi.org/10.1063/1.168744.

Guillaume Giudicelli, Alexander Lindsay, Logan Harbour, Casey Icenhour, Mengnan Li, Joshua E. Hansel, Peter German, Patrick Behne, Oana Marin, Roy H. Stogner, Jason M. Miller, Daniel Schwen, Yaqi Wang, Lynn Munday, Sebastian Schunert, Benjamin W. Spencer, Dewen Yushu, Antonio Recuero, Zachary M. Prince, Max Nezdyur, Tianchen Hu, Yinbin Miao, Yeon Sang Jung, Christopher Matthews, April Novak, Brandon Langley, Timothy Truster, Nuno Nobre, Brian Alger, David Andrš, Fande Kong, Robert Carlsen, Andrew E. Slaughter, John W. Peterson, Derek Gaston, and Cody Permann. 3.0 - MOOSE: Enabling massively parallel multiphysics simulations. *SoftwareX*, 26:101690, 2024. ISSN 2352-7110. doi: https://doi.org/10.1016/j.softx.2024.101690. URL https://www.sciencedirect.com/science/article/pii/S235271102400061X.

M. Hoelzl, G. T. A. Huijsmans, S. J. P. Pamela, M. Bécoulet, E. Nardon, F.J. Artola, B. Nkonga, C. V. Atanasiu, V. Bandaru, A. Bhole, D. Bonfiglio, A. Cathey, O. Czarny, A. Dvornova, T. Fehér, A. Fil, E. Franck, S. Futatani, M. Gruca, H. Guillard, J. W. Haverkort, I. Holod, D. Hu, S. K. Kim, S. Q. Korving, L. Kos, I. Krebs, L. Kripner, G. Latu, F. Liu, P. Merkel, D. Meshcheriakov, V. Mitterauer, S. Mochalskyy, J. A. Morales, R. Nies, N. Nikulsin, F. Orain, J. Pratt, R. Ramasamy, P. Ramet, C. Reux, K. Särkimäki, N. Schwarz, P. Singh Verma, S. F. Smith, C. Sommariva, E. Strumberger, D. C. van Vugt, M. Verbeek, E. Westerhof, F. Wieschollek, and J. Zielinski. The jorek non-linear extended mhd code and applications to large-scale instabilities and their control in magnetically confined fusion plasmas. *Nuclear Fusion*, 61(6):065001, 2021. doi: 10.1088/1741-4326/abf99f. URL https://dx.doi.org/10.1088/1741-4326/abf99f.

G. Danabasoglu, J.-F. Lamarque, J. Bacmeister, D. A. Bailey, A. K. DuVivier, J. Edwards, L. K. Emmons, J. Fasullo, R. Garcia, A. Gettelman, C. Hannay, M. M. Holland, W. G. Large, P. H. Lauritzen, D. M. Lawrence, J. T. M. Lenaerts, K. Lindsay, W. H. Lipscomb, M. J. Mills, R. Neale, K. W. Oleson, B. Otto-Bliesner, A. S. Phillips, W. Sacks, S. Tilmes, L. van Kampenhout, M. Vertenstein, A. Bertini, J. Dennis, C. Deser, C. Fischer, B. Fox-Kemper, J. E. Kay, D. Kinnison, P. J. Kushner, V. E. Larson, M. C. Long, S. Mickelson, J. K. Moore, E. Nienhouse, L. Polvani, P. J. Rasch, and W. G. Strand. The community earth system model version 2 (cesm2). *Journal of Advances in Modeling Earth Systems*, 12(2):e2019MS001916, 2020. doi: https://doi.org/10.1029/2019MS001916. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2019MS001916. e2019MS001916 2019MS001916.

J. N. Reddy. *Introduction to the Finite Element Method*. McGraw-Hill Education, New York, 3 edition, 2006. ISBN 9780072466850. URL https://www.accessengineeringlibrary.com/content/book/9780072466850.

Bengt Fornberg. *A Practical Guide to Pseudospectral Methods*. Cambridge University Press, Cambridge, UK, 1996.

David E Keyes, Lois C McInnes, Carol Woodward, William Gropp, Eric Myra, Michael Pernice, John Bell, Jed Brown, Alain Clo, Jeffrey Connors, Emil Constantinescu, Don Estep, Kate Evans, Charbel Farhat, Ammar Hakim, Glenn Hammond, Glen Hansen, Judith Hill, Tobin Isaac, Xiangmin Jiao, Kirk Jordan, Dinesh Kaushik, Efthimios Kaxiras, Alice Koniges, Kihwan Lee, Aaron Lott, Qiming Lu, John Magerlein, Reed Maxwell, Michael McCourt, Miriam Mehl, Roger Pawlowski, Amanda P Randles, Daniel Reynolds, Beatrice Rivière, Ulrich Rüde, Tim Scheibe, John Shadid, Brendan Sheehan, Mark Shephard, Andrew Siegel, Barry Smith, Xianzhu Tang, Cian Wilson, and Barbara Wohlmuth. Multiphysics simulations: Challenges and opportunities. *The International Journal of High Performance Computing Applications*, 27(1):4–83, 2013. doi: 10.1177/1094342012468181. URL https://doi.org/10.1177/1094342012468181.

J. A. K. Horwitz. Estimating the carbon footprint of computational fluid dynamics. *Physics of Fluids*, 36(4):045109, 04 2024. ISSN 1070-6631. doi: 10.1063/5.0199350. URL https://doi.org/10.1063/5.0199350.

Alexander Lavin, David Krakauer, Hector Zenil, Justin Gottschlich, Tim Mattson, Johann Brehmer, Anima Anandkumar, Sanjay Choudry, Kamil Rocki, Atılım Güneş Baydin, Carina Prunkl, Brooks Paige, Olexandr Isayev, Erik Peterson, Peter L. McMahon, Jakob Macke, Kyle Cranmer, Jiaxin Zhang, Haruko Wainwright, Adi Hanuka, Manuela Veloso, Samuel Assefa, Stephan Zheng, and Avi Pfeffer. Simulation intelligence: Towards a new generation of scientific methods. *arXiv:2112.03235*, 2021. doi: 10.48550/ARXIV.2112.03235. URL https://arxiv.org/abs/2112.03235.

Zongyi Li, Nikola Borislavov Kovachki, Kamyar Az-

izzadenesheli, Burigede liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=c8P9NQVtmnO.

Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=roNqYL0_XP.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, Mar 2021. ISSN 2522-5839. doi: 10.1038/s42256-021-00302-5. URL https://doi.org/10.1038/s42256-021-00302-5.

Yolanne Yi Ran Lee. Autoregressive renaissance in neural pde solvers, 2023. URL https://arxiv.org/abs/2310.19763.

Sergio Blanes, Fernando Casas, and Ander Murua. Splitting methods for differential equations, 2024. URL https://arxiv.org/abs/2401.01722.

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1): 79–87, 1991. doi: 10.1162/neco.1991.3.1.79.

George Em Karniadakis, Ioannis G. Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021. ISSN 2522-5820. doi: 10.1038/s42254-021-00314-5. URL https://doi.org/10.1038/s42254-021-00314-5.

Phillip Lippe, Bastiaan S. Veeling, Paris Perdikaris, Richard E Turner, and Johannes Brandstetter. PDE-refiner: Achieving accurate long rollouts with neural PDE solvers. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=Qv6468llWS.

Ameya D. Jagtap, Zhiping Mao, Nikolaus Adams, and George Em Karniadakis. Physics-informed neural networks for inverse problems in supersonic flows. *Journal of Computational Physics*, 466:111402, October 2022. ISSN 0021-9991. doi: 10.1016/j.jcp.2022.111402. URL http://dx.doi.org/10.1016/j.jcp.2022.111402.

Zhao Chen, Yang Liu, and Hao Sun. Physics-informed learning of governing equations from scarce data. *Nature Communications*, 12(1):6136, Oct 2021. ISSN 2041-1723. doi: 10.1038/s41467-021-26434-1. URL https://doi.org/10.1038/s41467-021-26434-1.

Yoeri Poels, Alessandro Pau, Christian Donner, Giulio Romanelli, Olivier Sauter, Cristina Venturini, Vlado Menkovski, the TCV Team, and the WPTE Team. Plasma state monitoring and disruption characterization using multimodal vaes. *Nuclear Fusion*, 65(9):096012, aug 2025. doi: 10.1088/1741-4326/adf121. URL https://dx.doi.org/10.1088/1741-4326/adf121.

Miles Cranmer. Interpretable machine learning for science with pysr and symbolicregression.jl, 2023. URL https://arxiv.org/abs/2305.01582.

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. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.10.045. URL https://www.sciencedirect.com/science/article/pii/S0021999118307125.

Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. Physics-informed neural operator for learning partial differential equations. *ACM / IMS J. Data Sci.*, 1 (3), may 2024. doi: 10.1145/3648506. URL https://doi.org/10.1145/3648506.

Shawn G Rosofsky, Hani Al Majed, and E A Huerta. Applications of physics informed neural operators. *Machine Learning: Science and Technology*, 4(2):025022, may 2023. doi: 10.1088/2632-2153/acd168. URL https://dx.doi.org/10.1088/2632-2153/acd168.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces with applications to pdes. *Journal of Machine Learning Research*, 24(89):1–97, 2023. URL http://jmlr.org/papers/v24/21-1524.html.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/69386f6bb1dfed68692a24c8686939b9-Paper.pdf.

Louis Serrano, Lise Le Boudec, Armand Kassaï Koupaï, Thomas X Wang, Yuan Yin, Jean-Noël

Vittaut, and patrick gallinari. Operator learning with neural fields: Tackling PDEs on general geometries. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=4jEjq5nhg1.

Anthony Zhou and Amir Barati Farimani. Predicting change, not states: An alternate framework for neural pde surrogates. *Computer Methods in Applied Mechanics and Engineering*, 441:117990, 2025. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2025.117990. URL https://www.sciencedirect.com/science/article/pii/S0045782525002622.

James Koch, Madelyn Shapiro, Himanshu Sharma, Draguna Vrabie, and Jan Drgona. Learning neural differential algebraic equations via operator splitting, 2025. URL https://arxiv.org/abs/2403.12938.

Benedikt Alkin, Andreas Fürst, Simon Schmid, Lukas Gruber, Markus Holzleitner, and Johannes Brandstetter. Universal physics transformers: A framework for efficiently scaling neural operators, 2024.

Michael McCabe, Bruno Régaldo-Saint Blancard, Liam Parker, Ruben Ohana, Miles Cranmer, Alberto Bietti, Michael Eickenberg, Siavash Golkar, Geraud Krawezik, Francois Lanusse, Mariel Pettee, Tiberiu Tesileanu, Kyunghyun Cho, and Shirley Ho. Multiple physics pretraining for physical surrogate models. In *NeurIPS 2023 AI for Science Workshop*, 2023a. URL https://openreview.net/forum?id=M12lmQKuxa.

Md Ashiqur Rahman, Robert Joseph George, Mogab Elleithy, Daniel Leibovici, Zongyi Li, Boris Bonev, Colin White, Julius Berner, Raymond A. Yeh, Jean Kossaifi, Kamyar Azizzadenesheli, and Anima Anandkumar. Pretraining codomain attention neural operators for solving multiphysics pdes, 2024.

Philipp Holl, Nils Thuerey, and Vladlen Koltun. Learning to control pdes with differentiable physics. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HyeSin4FPB.

Kiwon Um, Robert Brand, Yun, Fei, Philipp Holl, and Nils Thuerey. Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers, 2021. URL https://arxiv.org/abs/2007.00016.

Jonathan Citrin, Ian Goodfellow, Akhil Raju, Jeremy Chen, Jonas Degrave, Craig Donner, Federico Felici, Philippe Hamel, Andrea Huber, Dmitry Nikulin, David Pfau, Brendan Tracey, Martin Riedmiller, and Pushmeet Kohli. Torax: A fast and differentiable tokamak transport simulator in jax, 2024. URL https://arxiv.org/abs/2406.06718.

Kanishk Bhatia, Felix Koehler, and Nils Thuerey. Prdp: Progressively refined differentiable physics, 2025. URL https://arxiv.org/abs/2502.19611.

Francesca Bartolucci, Emmanuel de Bezenac, Bogdan Raonic, Roberto Molinaro, Siddhartha Mishra, and Rima Alaifari. Representation equivalent neural operators: a framework for alias-free operator learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=7LSEkvEGCM.

Qianying Cao, Somdatta Goswami, and George Em Karniadakis. Lno: Laplace neural operator for solving differential equations, 2023. URL https://arxiv.org/abs/2303.10528.

Tapas Tripura and Souvik Chakraborty. Wavelet neural operator for solving parametric partial differential equations in computational mechanics problems. *Computer Methods in Applied Mechanics and Engineering*, 404:115783, 2023. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2022.115783. URL https://www.sciencedirect.com/science/article/pii/S0045782522007393.

Olav Kallenberg. *Foundations of Modern Probability*. Probability and Its Applications. Springer, New York, NY, 1 edition, 1997. ISBN 978-0-387-22704-7. doi: 10.1007/b98838.

Vignesh Gopakumar, Stanislas Pamela, Lorenzo Zanisi, Zongyi Li, Ander Gray, Daniel Brennand, Nitesh Bhatia, Gregory Stathopoulos, Matt Kusner, Marc Peter Deisenroth, Anima Anandkumar, the JOREK Team, and MAST Team. Plasma surrogate modelling using fourier neural operators. *Nuclear Fusion*, 64(5):056025, 4 2024. doi: 10.1088/1741-4326/ad313a. URL https://dx.doi.org/10.1088/1741-4326/ad313a.

Michael McCabe, Peter Harrington, Shashank Subramanian, and Jed Brown. Towards stability of autoregressive neural operators. *Transactions on Machine Learning Research*, 2023b. ISSN 2835-8856. URL https://openreview.net/forum?id=RFfUUtKYOG.

Felix Koehler, Simon Niedermayr, rüdiger westermann, and Nils Thuerey. APEBench: A benchmark for autoregressive neural emulators of PDEs. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=iWc0qE116u.

N. Carey, L. Zanisi, S. Pamela, V. Gopakumar, J. Omotani, J. Buchanan, J. Brandstetter, F. Paischer, G. Galletti, and P. Setinek. Neural opera-

tor surrogate models of plasma edge simulations: feasibility and data efficiency, 2025. URL https://arxiv.org/abs/2502.17386.

Patrick Kidger. On neural differential equations, 2022. URL https://arxiv.org/abs/2202.02435.

Christopher Rackauckas, Yingbo Ma, Julius Martensen, Collin Warner, Kirill Zubov, Rohit Supekar, Dominic Skinner, Ali Ramadhan, and Alan Edelman. Universal differential equations for scientific machine learning, 2021. URL https://arxiv.org/abs/2001.04385.

Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural odes, 2019. URL https://arxiv.org/abs/1904.01681.

Patrick Kidger, James Foster, Xuechen Li, Harald Oberhauser, and Terry Lyons. Neural sdes as infinite-dimensional gans, 2021. URL https://arxiv.org/abs/2102.03657.

Michael Poli, Stefano Massaroli, Junyoung Park, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Graph neural ordinary differential equations, 2021. URL https://arxiv.org/abs/1911.07532.

Helge Holden, Kenneth Hvistendahl Karlsen, Knut-Andreas Lie, and Nils Henrik Risebro. *Splitting Methods for Partial Differential Equations with Rough Solutions: Analysis and MATLAB Programs*. EMS Series of Lectures in Mathematics. European Mathematical Society, Zürich, 2010. ISBN 978-3-03719-078-4. doi: 10.4171/078.

Lars Hörmander. *The analysis of linear partial differential operators I*, volume 256 of *Grundlehren der mathematischen Wissenschaften*. Springer, 1983. ISBN 3-540-12104-8. doi: 10.1007/978-3-642-96750-4.

Shev MacNamara and Gilbert Strang. *Operator Splitting*, pages 95–114. Springer International Publishing, Cham, 2016. ISBN 978-3-319-41589-5. doi: 10.1007/978-3-319-41589-5_3. URL https://doi.org/10.1007/978-3-319-41589-5_3.

Kimia Ghobadi, Nedialko S. Nedialkov, and Tamás Terlaky. On the discretize then optimize approach. Report 09T-005, McMaster University, Hamilton, Ontario, Canada, 2009.

Takashi Furuya, Michael Anthony Puthawala, Matti Lassas, and Maarten V. de Hoop. Can neural operators always be continuously discretized? In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=cyJxphdw3B.

Robert Joseph George, Jiawei Zhao, Jean Kossaifi, Zongyi Li, and Anima Anandkumar. Incremental spatial and spectral learning of neural operators for solving large-scale PDEs. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=xI6cPQObp0.

Jonas Actor, David T Fuentes, and Beatrice Riviere. Identification of kernels in a convolutional neural network: connections between the level set equation and deep learning for image segmentation. In Bennett A Landman and Ivana Išgum, editors, *Medical Imaging 2020: Image Processing*. SPIE, March 2020.

Boyang Chen, Claire E. Heaney, Jefferson L.M.A. Gomes, Omar K. Matar, and Christopher C. Pain. Solving the discretised multiphase flow equations with interface capturing on structured grids using machine learning libraries. *Computer Methods in Applied Mechanics and Engineering*, 426:116974, 2024a. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2024.116974. URL https://www.sciencedirect.com/science/article/pii/S0045782524002305.

Boyang Chen, Claire E. Heaney, and Christopher C. Pain. Using ai libraries for incompressible computational fluid dynamics, 2024b. URL https://arxiv.org/abs/2402.17913.

Vignesh Gopakumar, Ander Gray, Daniel Giles, Lorenzo Zanisi, Stanislas Pamela, Matt Kusner, and Marc Peter Deisenroth. Calibrated physics-informed uncertainty quantification, 2025. URL https://openreview.net/forum?id=cF6OoaYcRa.

David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. Learning factored representations in a deep mixture of experts, 2014. URL https://arxiv.org/abs/1312.4314.

Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=B1ckMDqlg.

Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5301–5310. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/rahaman19a.html.

Shaoxiang Qin, Fuyuan Lyu, Wenhui Peng, Dingyang Geng, Ju Wang, Xing Tang, Sylvie Leroyer, Naiping Gao, Xue Liu, and Liangzhu Leon Wang. Toward

a better understanding of fourier neural operators from a spectral perspective, 2024. URL https://arxiv.org/abs/2404.07200.

Siavash Khodakarami, Vivek Oommen, Aniruddha Bora, and George Em Karniadakis. Mitigating spectral bias in neural operators via high-frequency scaling for physical systems. *Neural Networks*, 193:108027, 2026. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2025.108027. URL https://www.sciencedirect.com/science/article/pii/S0893608025009074.

A. D. Polyanin and A. V. Manzhirov. *Handbook of Integral Equations*. CRC Press, Boca Raton, 1998. ISBN 0849328764.

Uri M. Ascher, Steven J. Ruuth, and Raymond J. Spiteri. Implicit-explicit runge-kutta methods for time-dependent partial differential equations. *Applied Numerical Mathematics*, 25:151–167, 1997. URL https://api.semanticscholar.org/CorpusID:14881937.

Jean Kossaifi, Nikola Kovachki, Zongyi Li, David Pitt, Miguel Liu-Schiaffini, Robert Joseph George, Boris Bonev, Kamyar Azizzadenesheli, Julius Berner, and Anima Anandkumar. A library for learning neural operators, 2024.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing. ISBN 978-3-319-24574-4.

Jayesh K Gupta and Johannes Brandstetter. Towards multi-spatiotemporal-scale generalized PDE modeling. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=dPSTDbGtBY.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021. URL https://arxiv.org/abs/2010.11929.

Maximilian Herde, Bogdan Raonic, Tobias Rohner, Roger Käppeli, Roberto Molinaro, Emmanuel de Bezenac, and Siddhartha Mishra. Poseidon: Efficient foundation models for PDEs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=JC1VKK3UXk.

Md Ashiqur Rahman, Zachary E. Ross, and Kamyar Azizzadenesheli. U-no: U-shaped neural operators, 2023. URL https://arxiv.org/abs/2204.11127.

Maarten V. de Hoop, Daniel Zhengyu Huang, Elizabeth Qian, and Andrew M. Stuart. The cost-accuracy trade-off in operator learning with neural networks, 2022. URL https://arxiv.org/abs/2203.13181.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Dochan Kwak and Cetin Kiris. Cfd for incompressible flows at nasa ames. *Computers & Fluids*, 38(3):504–510, 2009. ISSN 0045-7930. doi: https://doi.org/10.1016/j.compfluid.2008.06.010. URL https://www.sciencedirect.com/science/article/pii/S0045793008001965.

Peter Bauer, Alan Thorpe, and Gilbert Brunet. The quiet revolution of numerical weather prediction. *Nature*, 525(7567):47–55, Sep 2015. ISSN 1476-4687. doi: 10.1038/nature14956. URL https://doi.org/10.1038/nature14956.

Xiangdong Zhang, Li Luo, Ye Li, and Xiao-Chuan Cai. A fully implicit method for numerical simulation of blood flows with incompressible navier-stokes equations with slip and penetration boundary conditions. *Journal of Computational Physics*, 540:114267, 2025. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2025.114267. URL https://www.sciencedirect.com/science/article/pii/S0021999125005509.

C. Canuto, M.Y. Hussaini, A. Quarteroni, and T.A. Zang. *Spectral Methods: Evolution to Complex Geometries and Applications to Fluid Dynamics*. Scientific Computation. Springer Berlin Heidelberg, 2007. ISBN 9783540307280. URL https://books.google.co.uk/books?id=7COgEw5_EBQC.

Tai-Ping Liu. Shock waves for compressible navier-stokes equations are stable. *Communications on Pure and Applied Mathematics*, 39(5):565–594, 1986. doi: https://doi.org/10.1002/cpa.3160390502. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cpa.3160390502.

A. Hujeirat and R. Rannacher. A method for computing compressible, highly stratified flows in astrophysics based on operator splitting. *International Journal for Numerical Methods in Fluids*, 28(1):1–22, 1998. doi: https://doi.org/10.1002/(SICI)1097-0363(19980715)28:

1⟨1::AID-FLD690⟩3.0.CO;2-B. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291097-0363%2819980715%2928%3A1%3C1%3A%3AAID-FLD690%3E3.0.CO%3B2-B.

John Shebalin. Pseudospectral simulation of compressible turbulence using logarithmic variables. In *Proceedings of the 11th Computational Fluid Dynamics Conference*. AIAA, 1993. doi: 10.2514/6.1993-3375. URL https://arc.aiaa.org/doi/abs/10.2514/6.1993-3375.

Gordon Erlebacher, M. Y. Hussaini, H. O. Kreiss, and S. Sarkar. The analysis and simulation of compressible turbulence. *Theoretical and Computational Fluid Dynamics*, 2(2):73–95, 1990. ISSN 1432-2250. doi: 10.1007/BF00272136. URL https://doi.org/10.1007/BF00272136.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations, 2020. URL https://arxiv.org/abs/2003.03485.

Zongyi Li, Nikola Borislavov Kovachki, Chris Choy, Boyi Li, Jean Kossaifi, Shourya Prakash Otta, Mohammad Amin Nabian, Maximilian Stadler, Christian Hundt, Kamyar Azizzadenesheli, and Anima Anandkumar. Geometry-informed neural operator for large-scale 3d PDEs. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=86dXbqT5Ua.

Johannes Brandstetter, Daniel E. Worrall, and Max Welling. Message passing neural PDE solvers. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=vSix3HPYKSU.

Robert I. McLachlan and G. Reinout W. Quispel. Splitting methods. *Acta Numerica*, 11:341–434, 2002. doi: 10.1017/S0962492902000053.

Antonio Alguacil, Wagner Gonçalves Pinto, Michael Bauerheim, Marc C. Jacob, and Stéphane Moreau. Effects of boundary conditions in fully convolutional networks for learning spatio-temporal dynamics, 2021. URL https://arxiv.org/abs/2106.11160.

Michael McCabe, Payel Mukhopadhyay, Tanya Marwah, Bruno Regaldo-Saint Blancard, Francois Rozet, Cristiana Diaconu, Lucas Meyer, Kaze W. K. Wong, Hadi Sotoudeh, Alberto Bietti, Irina Espejo, Rio Fear, Siavash Golkar, Tom Hehir, Keiya Hirashima, Geraud Krawezik, Francois Lanusse, Rudy Morel, Ruben Ohana, Liam Parker, Mariel Pettee, Jeff Shen, Kyunghyun Cho, Miles Cranmer, and Shirley Ho. Walrus: A cross-domain foundation model for continuum dynamics, 2025.

Cesare Donati, Martina Mammarella, Giuseppe C. Calafiore, Fabrizio Dabbene, Constantino Lagoa, and Carlo Novara. A kernel-based approach to physics-informed nonlinear system identification, 2025. URL https://arxiv.org/abs/2509.07634.

Michael Zhang, Samuel Kim, Peter Y. Lu, and Marin Soljačić. Deep learning and symbolic regression for discovering parametric equations. *IEEE Transactions on Neural Networks and Learning Systems*, 35 (11):16775–16787, 2024. doi: 10.1109/TNNLS.2023.3297978.

W. E. Schiesser. *The Numerical Method of Lines*. Academic Press, 1991. ISBN 0-12-624130-9.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes]
2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes]
   (b) Complete proofs of all theoretical results. [Yes]
   (c) Clear explanations of any assumptions. [Yes]
3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [Yes]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Yes]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]
5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

Supplementary Materials

A Qualitative Comparison

Work (citation)	Physics-informed	Continuous-in-time	Modular/ Mixture of Experts	Interpretable	Extrapolation / Generalisation
[Li et al., 2021] (FNO)	✗	✗	✗	✗	✗
[Lu et al., 2021] (DeepONet)	✗	✓	✓	✗	✗
[Raissi et al., 2019] (PINNs)	✓	✓	✗	✓	✓
[Rackauckas et al., 2021] (UDE)	✓	✓	✓	✗	✗
[Li et al., 2024] (PINO)	✓	✗	✗	✗	✓
[Koch et al., 2025]	✓	✓	✓	✗	✓
[Zhou and Barati Farimani, 2025]	✗	✓	✗	✗	✗
[Donati et al., 2025]	✓	✗	✗	✓	✗
[Zhang et al., 2024]	✗	✓	✗	✓	✓
OpsSplit (Ours)	✓	✓	✓	✓	✓

Table 4: Comparison of different approaches for Neural PDE solvers. ✓: Yes, ✗: No, ✓: Partial.

Comparing across standard neural PDE solvers: Table 4 benchmarks the proposed OpsSplit framework against leading neural PDE solver methods, including FNO, DeepONet, and PINNs. We evaluate these methods across five critical dimensions: physical consistency, temporal continuity, architectural modularity, interpretability, and generalisation capability. As the comparison illustrates, existing approaches typically specialise in specific attributes at the expense of others; for instance, while PINNs provide physics-informed continuity, they lack the modular flexibility of Mixture of Experts (MoE) systems. Similarly, standard operator learners like FNO are efficient but struggle with interpretability and temporal flexibility. OpsSplit distinguishes itself by unifying these properties, offering a uniquely holistic framework that is simultaneously physics-informed, modular, and capable of robust extrapolation.

B Theoretical Analysis

Consider a PDE governing field variables $\mathbf{u} \in H^s(\Omega; \mathbb{R}^n)$:

$$\frac{\partial \mathbf{u}}{\partial t} = \mathcal{D}_X(\mathbf{u}) = \sum_{i=1}^{j} \lambda_i^l \, D_i^l(\mathbf{u}) + \sum_{i=1}^{k} \lambda_i^{nl} \, D_i^{nl}(\mathbf{u}), \qquad t \in [0, T]. \tag{24}$$

We approximate $\mathcal{D}_X$ using the OpsSplit formulation:

$$\widehat{\mathcal{D}}_X(\mathbf{u}) = \sum_{i=1}^{j} \lambda_i^l \, \mathrm{FD}_i(\mathbf{u}) + \sum_{i=1}^{k} \lambda_i^{nl} \, \mathrm{NO}_{\theta_i}(\mathbf{u}), \tag{25}$$

where FD_i are finite-difference operators and NO_{θ_i} are neural operators.

We measure errors in $H^{s'}(\Omega)$.

---

1. Assumptions

Assumption 1 (Regularity and Lipschitz Continuity)**.** *Each $D_i^{nl} : H^s \to H^{s'}$ is Lipschitz continuous on a compact set $K \subset H^s$, i.e.*

$$\|D_i^{nl}(\mathbf{u}) - D_i^{nl}(\mathbf{v})\|_{s'} \leq L_i \|\mathbf{u} - \mathbf{v}\|_s.$$

Assumption 2 (Approximation Classes)**.** *Let $\mathcal{N}(P)$ denote the class of neural operators with at most P parameters. Define the optimal approximation error:*

$$\epsilon_i^*(P) := \inf_{\theta:\, \mathrm{NO}_\theta \in \mathcal{N}(P)} \sup_{\mathbf{u} \in K} \|D_i^{nl}(\mathbf{u}) - \mathrm{NO}_\theta(\mathbf{u})\|_{s'}.$$

Assumption 3 (Additive Model Class)**.** *Define the monolithic class:*

$$\mathcal{N}_{\mathrm{mono}}(P) := \{\mathrm{NO}_\theta \in \mathcal{N}(P)\},$$

and the split class:

$$\mathcal{N}_{\mathrm{split}}(P) := \left\{ \sum_{i=1}^{k} \lambda_i^{nl} \mathrm{NO}_{\theta_i} : \sum_{i=1}^{k} P_i \leq P,\ \mathrm{NO}_{\theta_i} \in \mathcal{N}(P_i) \right\}.$$

Assumption 4 (Finite Difference Consistency)**.** *Each FD_i satisfies*

$$\|D_i^l(\mathbf{u}) - \mathrm{FD}_i(\mathbf{u})\|_{s'} \leq C_i h^{p_i} \|\mathbf{u}\|_{s+p_i}.$$

---

2. Capacity Allocation Lower Bound

Lemma 1 (Capacity Allocation Lower Bound)**.** *Let $\mathcal{D}^{nl} = \sum_{i=1}^{k} \lambda_i^{nl} D_i^{nl}$. Then:*

$$\inf_{\mathrm{NO}_\theta \in \mathcal{N}(P)} \sup_{\mathbf{u} \in K} \|\mathcal{D}^{nl}(\mathbf{u}) - \mathrm{NO}_\theta(\mathbf{u})\|_{s'} \ \geq \ \inf_{\{P_i\}: \sum P_i \leq P} \sum_{i=1}^{k} |\lambda_i^{nl}| \, \epsilon_i^*(P_i). \tag{26}$$

Proof. Let $\mathrm{NO}_\theta \in \mathcal{N}(P)$ be arbitrary. For each i, define the residual operator:

$$R_i(\mathbf{u}) := D_i^{nl}(\mathbf{u}) - \widetilde{D}_i(\mathbf{u}),$$

where $\widetilde{D}_i$ denotes the implicit contribution of NO_θ toward approximating D_i^{nl}.

Since NO_θ has finite capacity P, its approximation of the sum must implicitly allocate representational capacity across the components. This induces an effective decomposition into sub-approximations with budgets $\{P_i\}$ satisfying $\sum P_i \leq P$.

By definition of ϵ_i^*, each component satisfies:

$$\sup_{\mathbf{u}\in K} \|D_i^{nl}(\mathbf{u}) - \widetilde{D}_i(\mathbf{u})\|_{s'} \geq \epsilon_i^*(P_i).$$

Applying the triangle inequality,

$$\|\mathcal{D}^{nl} - \mathrm{NO}_\theta\|_{s'} \geq \sum_{i=1}^{k} |\lambda_i^{nl}| \|D_i^{nl} - \widetilde{D}_i\|_{s'}.$$

Taking the supremum over $\mathbf{u}$ and infimum over θ yields the result. □

---

3. Approximation Error Bounds

Theorem 2 (OpsSplit Error Bound)**.** *Under the stated assumptions,*

$$\epsilon_{\mathrm{OpS}}(P) \leq \delta_{\mathrm{lin}} + \inf_{\{P_i\}:\sum P_i \leq P} \sum_{i=1}^{k} |\lambda_i^{nl}|\, \epsilon_i^*(P_i), \tag{27}$$

where

$$\delta_{\mathrm{lin}} := \sum_{i=1}^{j} |\lambda_i^l|\, C_i h^{p_i} \|\mathbf{u}\|_{s+p_i}.$$

Proof. By the triangle inequality,

$$\|\mathcal{D}_X - \widehat{\mathcal{D}}_X\|_{s'} \leq \sum_{i=1}^{j} |\lambda_i^l| \|D_i^l - \mathrm{FD}_i\|_{s'} + \sum_{i=1}^{k} |\lambda_i^{nl}| \|D_i^{nl} - \mathrm{NO}_{\theta_i}\|_{s'}.$$

The first term is bounded by δ_{lin} via Assumption 4.

Optimizing over $\{\theta_i\}$ subject to $\sum P_i \leq P$ yields the second term. □

Theorem 3 (Comparison with Monolithic Models)**.** *Let $\epsilon_{\mathrm{mono}}(P)$ denote the optimal error over $\mathcal{N}_{\mathrm{mono}}(P)$. Then:*

$$\epsilon_{\mathrm{mono}}(P) \;\geq\; \inf_{\{P_i\}:\sum P_i \leq P} \sum_{i=1}^{k} |\lambda_i^{nl}|\, \epsilon_i^*(P_i). \tag{28}$$

Proof. This follows directly from Lemma 1. □

---

4. Out-of-Distribution Generalization

Theorem 4 (Coefficient Generalization)**.** *Let $\boldsymbol{\lambda}, \boldsymbol{\lambda}' \in \mathbb{R}^k$. Then the OpsSplit error satisfies:*

$$\|\mathcal{D}_X^{\boldsymbol{\lambda}'}(\mathbf{u}) - \widehat{\mathcal{D}}_X^{\boldsymbol{\lambda}'}(\mathbf{u})\|_{s'} \leq \delta_{\mathrm{lin}}(\boldsymbol{\lambda}') + \sum_{i=1}^{k} |(\lambda_i^{nl})'|\, \delta_i, \tag{29}$$

where δ_i is the training error of NO_{θ_i}.

Proof. Directly,

$$\mathcal{D}_X^{\boldsymbol{\lambda}'} - \widehat{\mathcal{D}}_X^{\boldsymbol{\lambda}'} = \sum_{i=1}^{j} (\lambda_i^l)'(D_i^l - \mathrm{FD}_i) + \sum_{i=1}^{k} (\lambda_i^{nl})'(D_i^{nl} - \mathrm{NO}_{\theta_i}).$$

Apply the triangle inequality and definitions of δ_{lin} and δ_i. □

---

5. Temporal Stability

Corollary 5 (Euler Stability)**.** *Let L be the Lipschitz constant of $\mathcal{D}_X$. Then under explicit Euler:*

$$\|\mathbf{u}^N - \hat{\mathbf{u}}^N\|_s \leq \frac{e^{LN\Delta t} - 1}{L} \left(\delta_{\mathrm{lin}} + \sum_{i=1}^{k} |\lambda_i^{nl}| \delta_i + \mathcal{O}(\Delta t) \right). \tag{30}$$

Proof. Standard discrete Grönwall argument applied to the error recursion. □

---

6. Transferability

Proposition 6 (Operator Transfer)**.** *Let systems A and B differ only in operators $D_i^{nl,A}$ and $D_i^{nl,B}$. Then:*

$$\delta_i^{(B)} \leq \delta_i^{(A)} + \sup_{\mathbf{u} \in K_B} \|D_i^{nl,B}(\mathbf{u}) - D_i^{nl,A}(\mathbf{u})\|_{s'}. \tag{31}$$

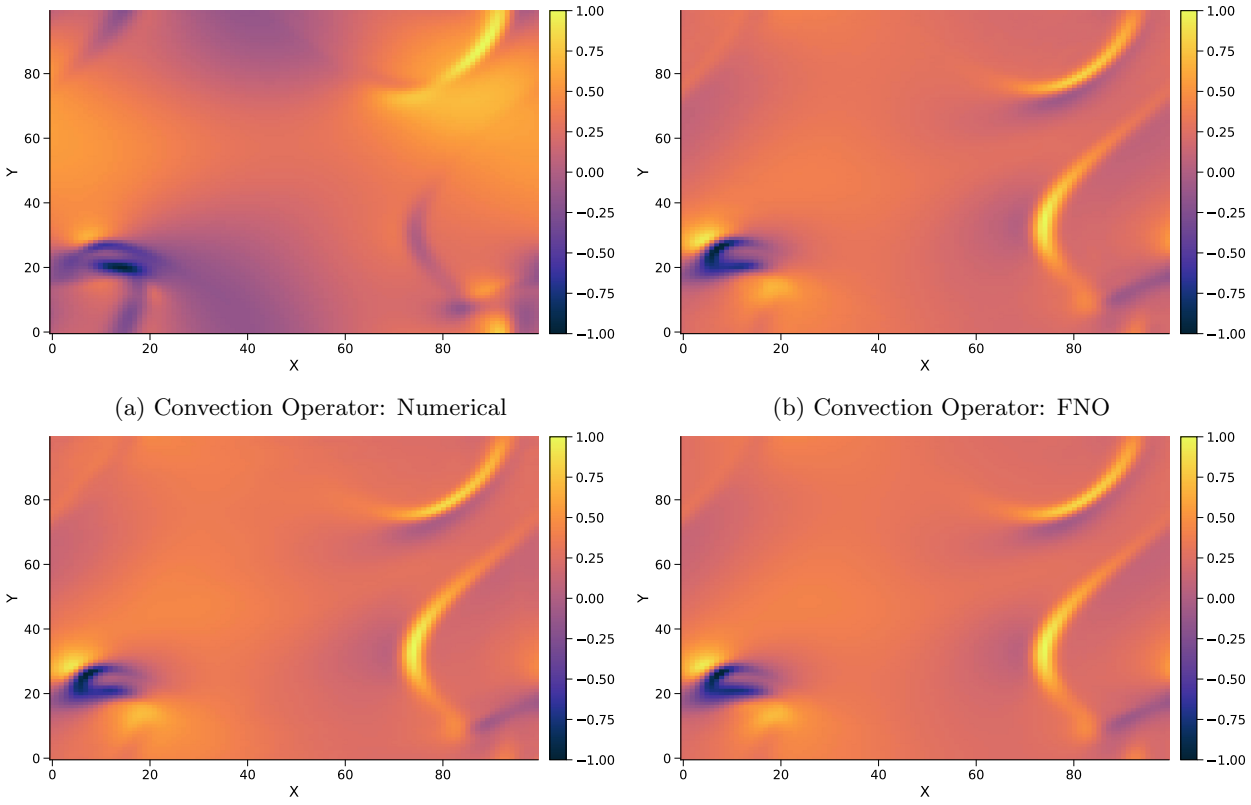
Proof. Immediate from the triangle inequality. □

C Interpretability

As discussed in section 5.1, constructing neural operators to explicitly learn physical operators within the OpsSplit framework enables soft enforcement of PDE constraints during prediction. This approach yields models that demonstrate improved fitting and enhanced capability for temporal extrapolation and generalisation to unseen physical conditions. Moreover, the learned physical operators provide inherent interpretability, as their behaviour can be directly verified against numerical estimations of the corresponding operators. This interpretability facilitates identification of failure modes within the prediction mechanism and enables more parameter-efficient, physics-aware model debugging strategies.

C.1 Incompressible Navier-Stokes

Figure 7 visualises the convection operator employed in the numerical solver (fig. 7a) alongside those learned by various neural operator architectures: FNO (fig. 7b), CNO (fig. 7c), and UNO (fig. 7d). Each neural operator successfully captures the nuances and general behaviour of the convection operator, approximating the underlying gradients to varying degrees of fidelity, as evidenced by the variations across architectures. These visualisations constitute a qualitative interpretability study wherein the learned convection operators, evaluated in latent space, are compared against their numerical counterparts computed in physical space. All operators are normalised to the range $[-1, 1]$ to facilitate visual comparison. The observation that distinct operator architectures learn similar attributes of the convection operator warrants further investigation in future work.



(a) Convection Operator: Numerical

(b) Convection Operator: FNO

(c) Convection Operator: CNO

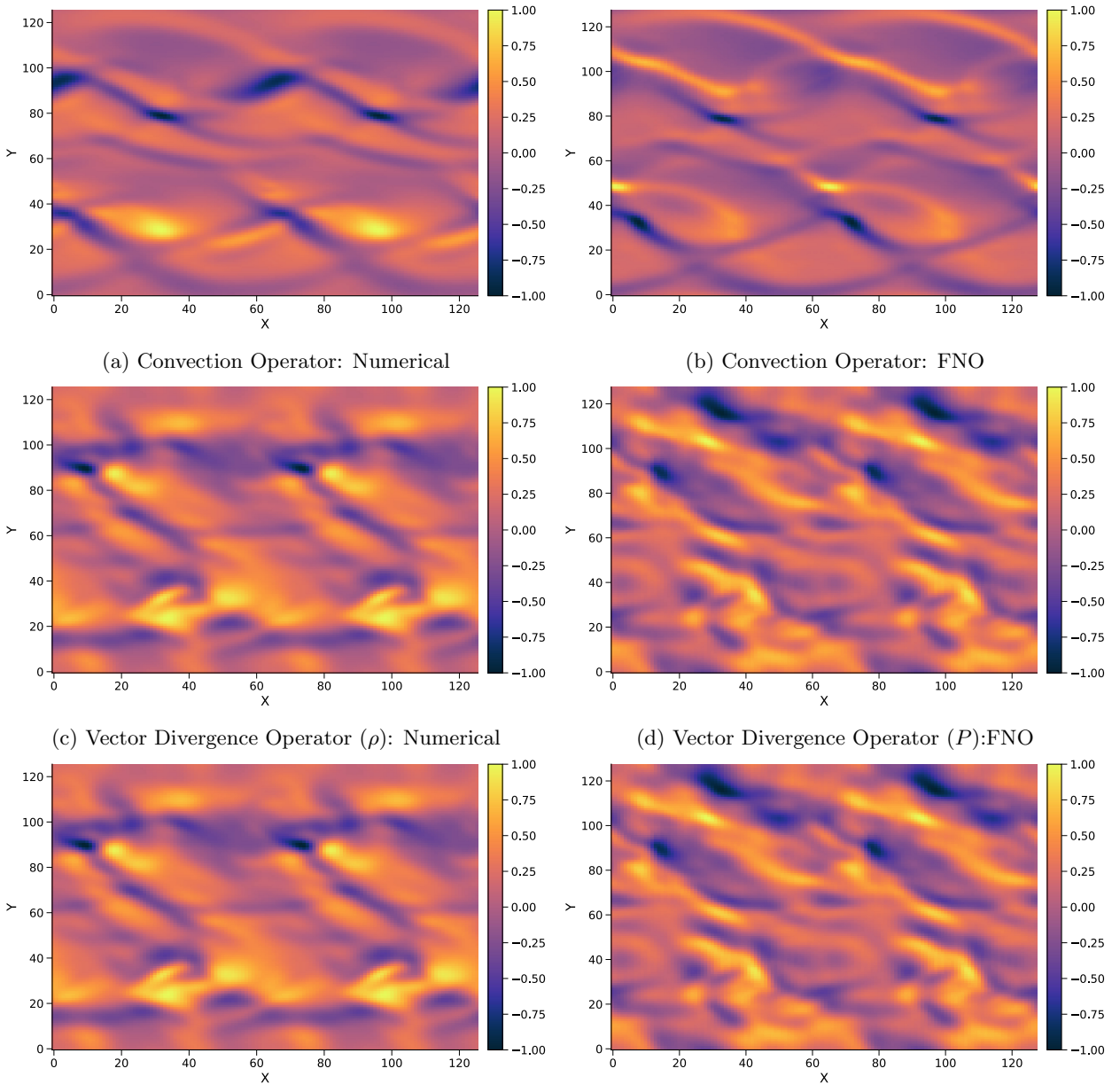
(d) Convection Operator: UNO

Figure 7: Incompressible Navier–Stokes: comparing across convection operator obtained numerically against those learnt using various neural operators.

C.2 Compressible Navier-Stokes

Figure 8 visualises both the convection and vector divergence operators computed via finite differences (figs. 8a, 8c and 8e) alongside their learned counterparts from an FNO architecture (figs. 8b, 8d and 8f, respectively).

For the convection operator, the FNO successfully captures the general fluid motion across the domain, though it fails to reproduce certain fine-scale features present in the physical operator. As formulated in eqs. (20) and (22), the vector divergence operator characterises the advection of scalar fields, and a single neural operator learns its effects on both pressure and density. Figure 8 demonstrates that while the FNO captures dominant features, it introduces spurious characteristics into the learned operation. This suggests the need for further investigation into alternative operator splitting strategies. As with the incompressible case, these visualisations provide qualitative interpretability analysis, with learned operators evaluated in latent space and numerical operators displayed in physical space, all normalised to $[-1, 1]$ for effective visual comparison.



(a) Convection Operator: Numerical

(b) Convection Operator: FNO

(c) Vector Divergence Operator (ρ): Numerical

(d) Vector Divergence Operator (P):FNO

(e) Vector Divergence Operator (P): Numerical

(f) Vector Divergence Operator (P): FNO

Figure 8: Compressible Navier–Stokes: comparing across convection and vector divergence operator obtained numerically against those learnt using various neural operators.

D Operator Splitting: A Guideline

Operator splitting, in its classical numerical sense, refers to *temporal* (fractional-step) methods that advance each sub-operator sequentially over separate sub-intervals—for example, Lie–Trotter or Strang splitting—introducing splitting errors that arise from the non-commutativity of the split operators [Holden et al., 2010, MacNamara and Strang, 2016]. The OpsSplit framework described in section 4 adopts a conceptually related but structurally distinct strategy: an *additive spatial decomposition* following the Method of Lines (MoL) paradigm [Schiesser, 1991]. Here the spatial operator $D_X(u)$ is decomposed into a sum of physically motivated sub-operators, each evaluated *simultaneously* on the current state u^n, and their combined contribution constitutes the right-hand side (RHS) of a time-continuous ODE:

$$\frac{\mathrm{d}u}{\mathrm{d}t} = \sum_{i=1}^{j} \lambda_{l_i} \mathbb{FD}_i(u) + \sum_{i=1}^{k} \lambda_{nl_i} \mathbb{NO}_i(u), \tag{32}$$

which is then integrated forward using a standard ODE solver (e.g., explicit Euler or Runge–Kutta). Crucially, *no sequential fractional time steps are executed*: all sub-operators see the same state u^n at each integration point, and their outputs are summed before the time-update is applied. Consequently, the temporal non-commutativity errors and intermediate fractional boundary conditions that characterise classical Strang splitting are *not* present in this framework. The approximation error instead manifests in the RHS itself—from replacing continuous spatial operators with finite-difference stencils ($\mathbb{FD}$) and learned neural operators ($\mathbb{NO}$)—and is fully decoupled from the temporal integration error, which is controlled independently by the choice of ODE solver.

The practical motivation for the decomposition in eq. (32) mirrors that of classical splitting: isolating physical processes (e.g., advection from diffusion) allows each to be approximated by the most suitable tool—fixed stencils for well-understood linear physics and neural operators for complex non-linear phenomena—while preserving a monolithic time integration pipeline. This section provides guidelines for designing effective additive decompositions within this neural Method of Lines framework.

D.1 Implementation Guidelines

Since no universal decomposition rule exists [McLachlan and Quispel, 2002], empirical validation remains essential. For the additive MoL framework, a robust decomposition strategy should adhere to the following principles:

1. **Physics-driven process identification:** Identify dominant physical processes and separate terms by mathematical character—linearity versus nonlinearity, locality versus non-locality. Assign linear, analytically tractable operators (e.g., diffusion, linear advection) to fixed finite-difference stencils, and reserve neural operator capacity for operators whose mathematical form is complex or whose effective behaviour is difficult to encode analytically (e.g., non-linear convection, coupled pressure–velocity effects from the pressure Poisson equation).

2. **Simultaneous evaluation consistency:** Because all sub-operators are evaluated on the same state u^n in the additive framework, each $\mathbb{NO}_i$ or $\mathbb{FD}_i$ must be designed to act on the *full* current state, not an intermediate or partially-advanced state. This distinguishes OpsSplit from fractional-step methods where later sub-operators receive states that have already been advanced by earlier sub-operators.

3. **Timescale and stiffness management:** Operators with disparate timescales (e.g., fast diffusion versus slow advection) may impose restrictive time-step requirements via the CFL condition when treated fully explicitly. Within the MoL framework, stiff linear operators can be incorporated via implicit-explicit (IMEX) integrators [Ascher et al., 1997], treating the $\mathbb{FD}$ terms implicitly and the $\mathbb{NO}$ terms explicitly, without altering the additive spatial decomposition.

4. **Conservation and invariant preservation:** Where physical conservation laws (mass, momentum, energy) or geometric constraints (e.g., divergence-free velocity fields) are critical, the decomposition should be designed so that each neural operator can learn to enforce the relevant invariants in its output on the shared state u^n. Conservation can be monitored via the physics residual error during training and rollout (fig. 4), providing an interpretable diagnostic tied directly to each operator's individual contribution to the RHS.

5. **Modular reusability:** Because sub-operators in the additive formulation share no sequential coupling through intermediate states, they can be trained and validated independently before being composed into the full RHS sum. This modularity supports transfer across related PDE families: a neural operator trained to approximate the convection operator for one set of PDE parameters can be fine-tuned or directly re-used for a related system, as demonstrated in appendix I.

D.2 Additive Decomposition and Approximation Error

In the additive RHS formulation of eq. (32), the forward Euler discrete update takes the form

$$u^{n+1} = u^n + \Delta t \left(\sum_{i=1}^{j} \lambda_{l_i} \, \mathbb{FD}_i(u^n) + \sum_{i=1}^{k} \lambda_{nl_i} \, \mathbb{NO}_i(u^n) \right), \tag{33}$$

which is identical to eq. (11) in the main text. Higher-order ODE integrators (e.g., explicit Runge–Kutta methods) may be substituted without altering the additive spatial structure, improving temporal accuracy independently of the decomposition choice.

The dominant error sources in this framework are therefore:

1. **Spatial approximation error from $\mathbb{FD}$:** Each finite-difference stencil $\mathbb{FD}_i$ introduces a truncation error that depends on the stencil order. A second-order stencil yields a spatial error of $\mathcal{O}(h^2)$ in the grid spacing h; higher-order stencils reduce this but may degrade out-of-distribution generalisation, as evidenced in table 5.

2. **Neural operator approximation error from $\mathbb{NO}$:** Each neural operator $\mathbb{NO}_i$ introduces a function approximation error bounded by its capacity and training distribution. Because each $\mathbb{NO}_i$ is trained to approximate a specific physical operator acting on individual snapshots—rather than on intermediate partially-advanced states—the error is localised to the RHS evaluation at each time step and does not accumulate across fractional sub-steps.

3. **Temporal integration error:** For explicit Euler the global temporal error is $\mathcal{O}(\Delta t)$; substituting a fourth-order Runge–Kutta scheme reduces this to $\mathcal{O}(\Delta t^4)$. This error is orthogonal to the spatial decomposition and is controlled independently by the ODE solver.

There is no splitting error analogous to Strang or Lie–Trotter schemes because the sub-operators are *summed*, not sequentially composed. The commutator $[\mathcal{A}, \mathcal{B}]$ is therefore not a direct source of error in this framework; non-commutativity of spatial operators affects the RHS only to the extent that each $\mathbb{FD}_i$ or $\mathbb{NO}_i$ must individually capture the correct operator action on the current state u^n. This property simplifies both the error analysis and the training procedure: operators can be trained and validated independently without reasoning about their sequential interaction.

D.3 Boundary Conditions

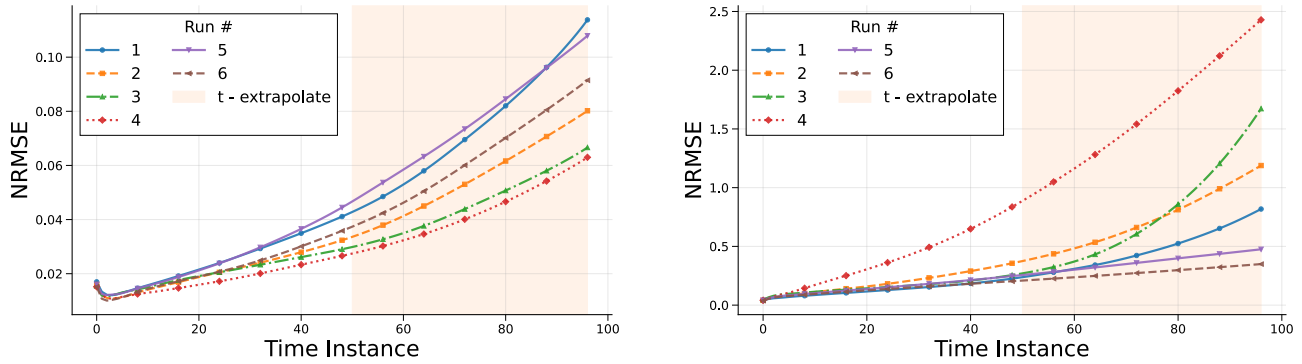
Boundary conditions (BCs) are defined for the complete operator D_X and must be respected by each sub-operator in the additive sum. In the MoL formulation of eq. (32), BCs are applied once to the full state u^n before the RHS is evaluated; there is no need to prescribe intermediate fractional boundary states as required in sequential fractional-step methods. Each $\mathbb{FD}_i$ and $\mathbb{NO}_i$ is therefore designed to operate on states that already satisfy the global BCs. In the experiments reported here, periodic BCs are exploited through standard circular padding in both stencil convolutions and neural operator architectures. Extension to non-periodic BCs (e.g., Dirichlet or Neumann) can be achieved by augmented padding schemes [Alguacil et al., 2021, McCabe et al., 2025].

D.4 Neural Extension

The MoL perspective clarifies how classical splitting intuition translates to the neural setting. Practitioners should leverage domain expertise for physics-driven decomposition, exploiting modular Mixture of Experts (MoE) structures to experiment with different operator groupings efficiently. Individual neural operators should be validated in isolation before being combined into the full RHS, taking advantage of the decoupled error structure

identified in appendix D.2. Transfer learning across related physical systems is further facilitated by the additive independence of sub-operators (see appendix I). Conservation laws and PDE residuals should be monitored during rollout to detect failure modes localised to specific physical operators, supporting interpretable model diagnosis.

When devising decompositions for neural architectures, the key trade-off remains between physical fidelity and computational resource constraints. Deploying distinct neural operators for individual physical processes increases model parameterisation and memory overhead. To mitigate this, the splitting strategy should prioritise identifying vector operations that are invariant across the target family of PDEs, thereby reducing redundant parameterisation. As demonstrated in appendix I, neural operators exhibit strong generalisation capabilities across different regimes; consequently, the design process should emphasise the development of reusable, modular operator models that can be transferred across related physical systems, maximising the return on computational investment.



(a) in-distribution

(b) out-of-distribution

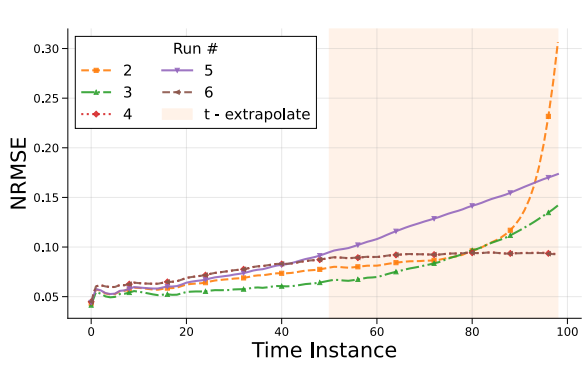
Figure 9: Incompressible Navier–Stokes: Rollout Error of the the various OpsSplit ablations as described in table 5. We experiment each ablation across both in and out-of-distribution. While we notice that higher order finite-difference stencils deployed to model the linear operators perform best within distribution, they fail to extend this to outside distribution. Learnt approximations of the linear diffusion operator either using a linear convolutional operator (5) or a neural operator (6) allows for better generalisation.

In table 5, we experiment with different architectures deployed within our OpSplit method. $FD(n)$ represents a fixed finite difference kernel of order n, Linear corresponds to a learnable linear convolution, and NO represents an FNO. We observe that higher-order finite-difference stencils modelling linear operators perform best within the distribution, but fail to generalise effectively to out-of-distribution contexts. Learnt approximations of the linear diffusion operator, whether using a linear convolutional operator (5) or a neural operator (6), enable superior generalisation for the PDE. This is further validated within the temporal rollout plots in fig. 9.

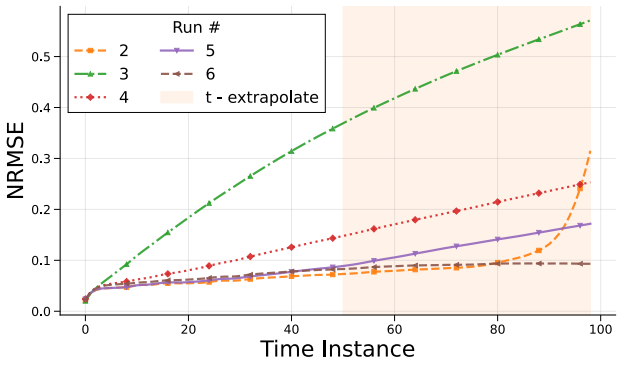
Ablation	Convection	Diffusion		Train Time	NRMSE			
	$(\mathbf{v}\cdot\nabla)\mathbf{v}$	$\nabla^2\mathbf{v}$	Parameters	(hrs:mins)	Test	t-extrapolate	OOD	OOD+t-extrapolate
1	NO	FD (2)	13437321	1:11	0.0270	0.0796	0.1471	0.4731
2	NO	FD (4)	13437339	1:12	0.0244	0.0600	0.1993	0.6995
3	NO	FD (6)	13437350	1:13	0.0221	**0.0504**	0.1576	0.5303
4	NO	FD (8)	13437378	1:13	**0.0214**	0.0568	0.4195	1.5992
5	NO	Linear	13438196	1:13	0.0283	0.0823	0.1581	0.3693
6	NO	NO	26874628	2:18	0.0241	0.0670	**0.1385**	**0.2815**

Table 5: Incompressible Navier–Stokes: Performance comparison across different methods of operator splitting and subsequent operator choices. $FD(n)$ represents a fixed finite difference kernel of order n, Linear corresponds to a learnable linear convolution, and NO represents an FNO.

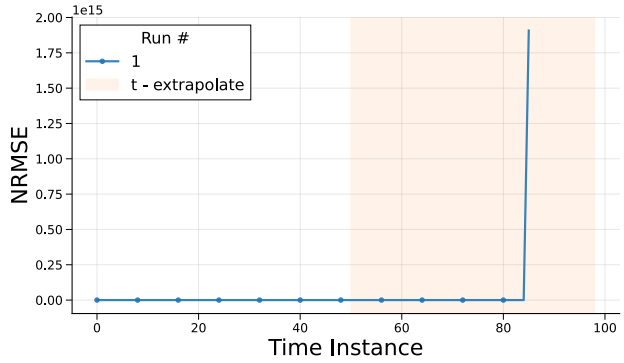
We performed a similar ablation of OpsSplit strategies across the physical operators within Compressible Navier-Stokes, as outlined in Section 5.2, where NO_i represents the i^{th} neural operator within the system. We observe that using a fixed linear operator over nonlinear operators tends to result in instability, leading to divergent



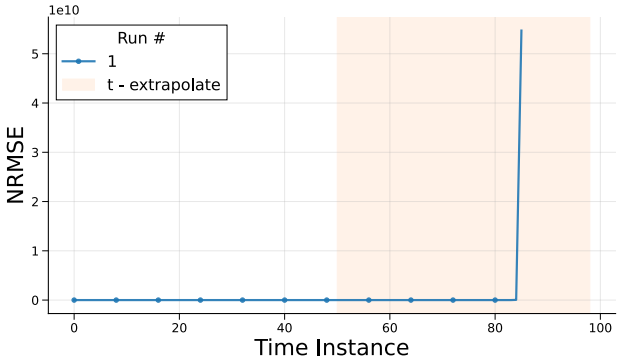
(a) in-distribution



(b) out-of-distribution



(c) Training Times



(d) Training Times

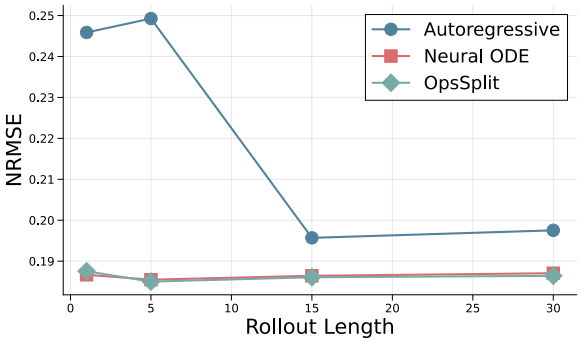
Figure 10: Compressible Navier–Stokes: Rollout Error of the the various OpsSplit ablations as described in table 6. We experiment each ablation across both in and out-of-distribution. We notice that when we approximate nonlinear operators using FD stencils the prediction blows up when extrapolated in time. The general trend we notice is that approximating using neural operators offer better stability both in and out-of-distribution.

predictions when extrapolating further in time. The general trend suggests that employing neural operators within the splitting strategies offers better stability both in and out of distribution. As indicated in the table, these performance improvements come at the cost of increased parameterisation and/or training time. We also note that using explicitly defined fixed kernels offers better performance than using learnable linear operators. Figure 10 demonstrate the temporal rollout of error for each OpsSplit strategy.

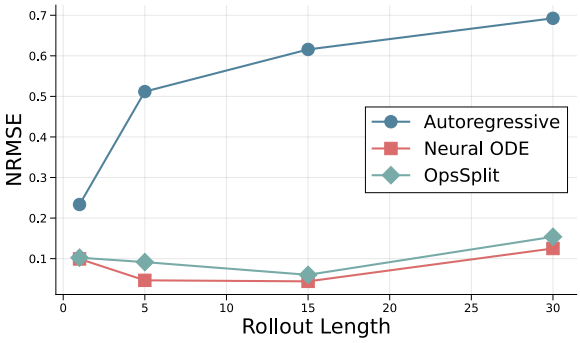
Ablation	Continuity	Convection	Pressure Gradient	Pressure Advection	Pressure Expansion		Train Time	NRMSE			
	$\nabla \cdot (\rho \mathbf{v})$	$(\mathbf{v} \cdot \nabla)\mathbf{v}$	$\nabla \mathbf{P}$	$\mathbf{v} \cdot \nabla P$	$P(\nabla \cdot \mathbf{v})$	Parameters	(hrs:mins)	Test	t-extrapolate	OOD	OOD+t-extrapolate
1	NO_1	NO_2	FD (4)	FD (4)	FD (4)	26874627	3:56	0.0824	-	0.0851	-
2	NO_1	NO_2	FD (4)	FD (4)	NO_3	40311940	5:43	0.0762	0.1441	**0.0785**	0.1564
3	NO_1	NO_2	Linear	Linear	Linear	26874666	3:52	**0.0639**	**0.1115**	0.2061	0.4201
4	NO_1	NO_2	Linear	Linear	NO_3	40311979	5:41	0.0866	0.1177	0.1329	0.2839
5	NO_1	NO_2	FD (4)	NO_1	NO_1	26874627	5:39	0.0822	0.1691	0.0852	0.1812
6	NO_1	NO_1	FD (4)	NO_1	NO_1	13437313	7:29	0.0865	0.1173	0.0887	**0.1252**

Table 6: Compressible Navier–Stokes: Performance comparison across different methods of operator splitting and subsequent operator choices. $FD(n)$ represents a fixed finite difference kernel of order n, Linear corresponds to a learnable linear convolution, and NO represents an FNO. NO_i represents the i^{th} neural operator within the system.

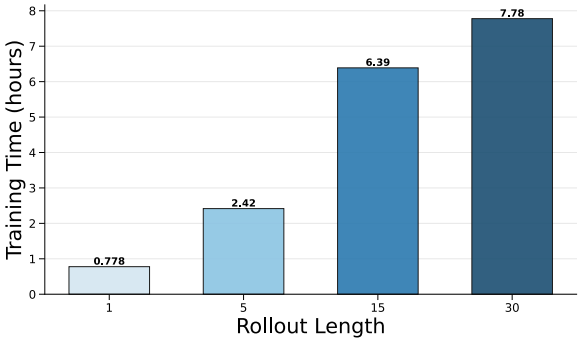
E Rollout Length



(a) in-distribution



(b) out-of-distribution

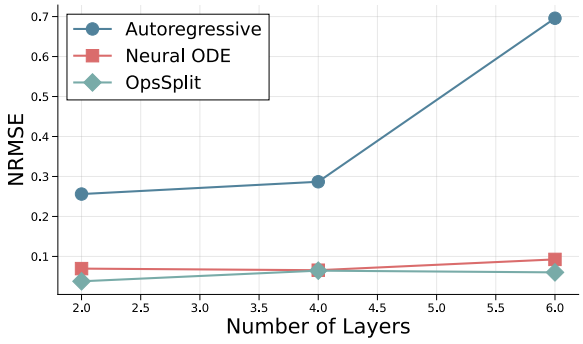


(c) Training Times

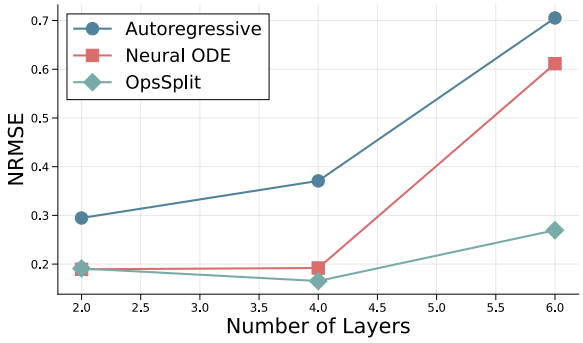
Figure 11: Incompressible Navier–Stokes: Rollout length of the FNO deployed across each method for both in and out-of-distribution.

As demonstrated by Koehler et al. [2024], training autoregressive neural PDE models with longer rollout lengths yields improved temporal stability. Our ablation studies corroborate this finding, with fig. 11a showing that extended rollout lengths enhance the performance of autoregressive models. In contrast, temporally continuous methods, specifically neural ODEs and our OpsSplit approach, exhibit marginal sensitivity to rollout length for both in-distribution and out-of-distribution scenarios. This insensitivity enables more memory and computationally efficient training paradigms. Notably, for out-of-distribution evaluation (fig. 11b), autoregressive models demonstrate degraded performance with longer rollouts, likely attributable to the increasing divergence between novel physics and the training distribution as temporal evolution progresses.

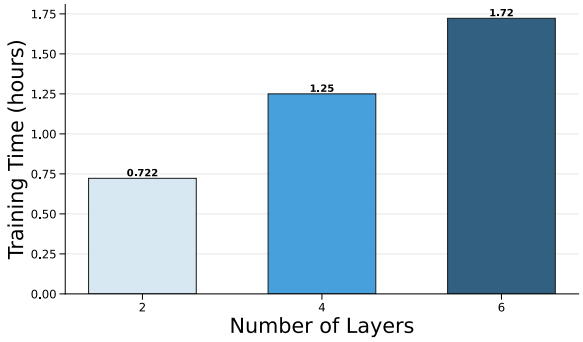
F Model Efficiency



(a) in-distribution



(b) out-of-distribution

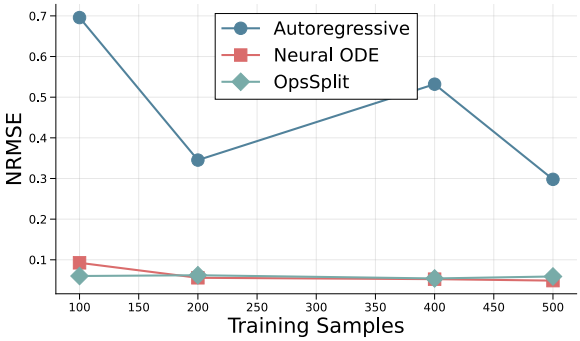


(c) Training Times

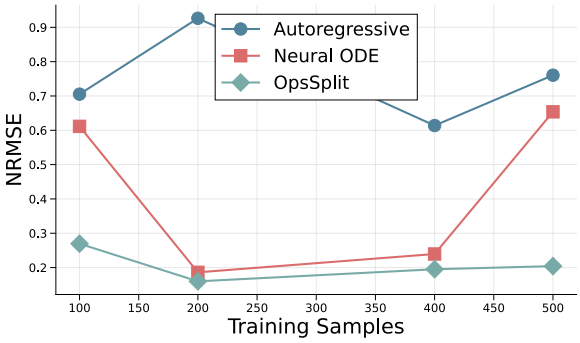
Figure 12: Incompressible Navier–Stokes: Data Efficiency of the FNO deployed across each method for both in and out-of-distribution.

Figure 12 demonstrates that the OpsSplit method exhibits exceptional parameter efficiency, achieving comparable performance across varying FNO depths. Conversely, the autoregressive approach shows deteriorating performance with increased model capacity, likely due to optimisation challenges in high-dimensional parameter spaces that demand greater computational resources to locate satisfactory minima. Across all model sizes, OpsSplit consistently outperforms competing methods for both in-distribution and out-of-distribution evaluation scenarios.

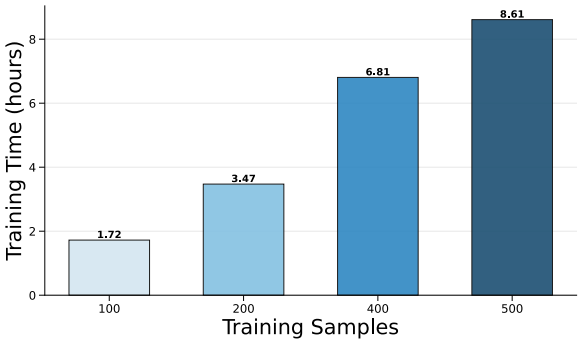
G Data Efficiency



(a) in-distribution



(b) out-of-distribution

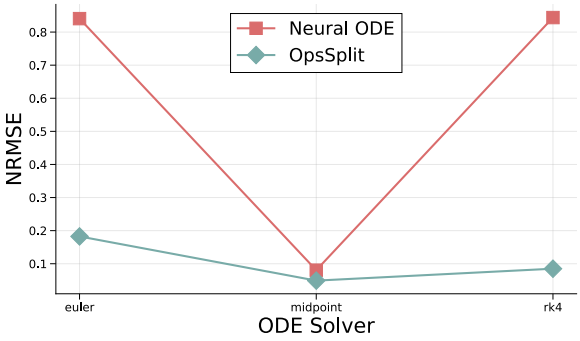


(c) Training Times

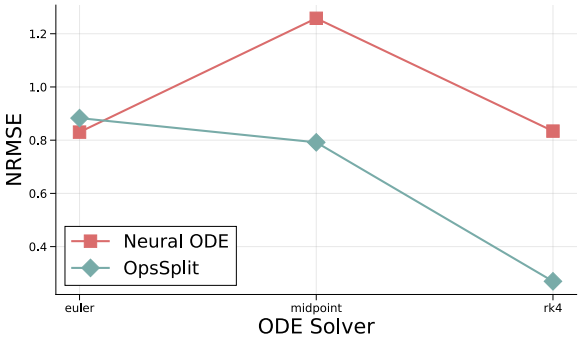
Figure 13: Incompressible Navier–Stokes: Data Efficiency of the FNO deployed across each method for both in and out-of-distribution.

Figure 13 demonstrates that the OpsSplit method provides a data-efficient training paradigm, exhibiting minimal sensitivity to dataset size variations. While the neural ODE approach achieves comparable performance on in-distribution data, it lacks explicit PDE structure encoding and consequently struggles to generalise to novel physical regimes. The autoregressive method benefits from additional training data, showing monotonic improvement with dataset size. However, OpsSplit consistently achieves superior performance with substantially reduced data requirements compared to baseline approaches.

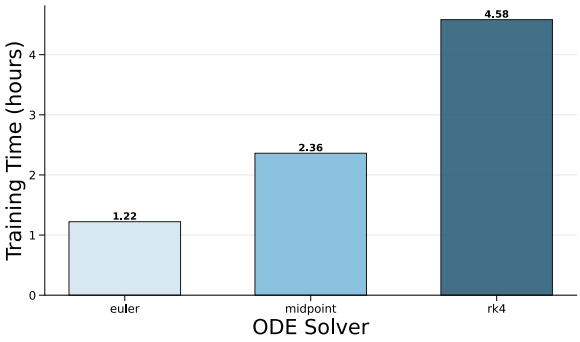
H Temporal Integration Method



(a) in-distribution



(b) out-of-distribution

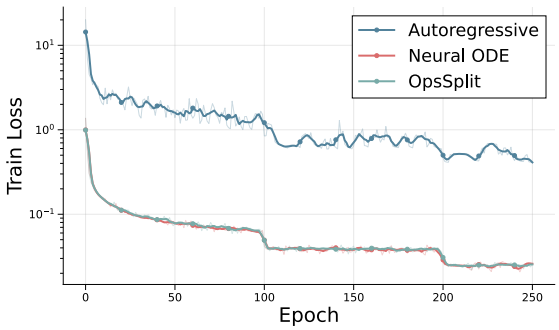


(c) Training Times

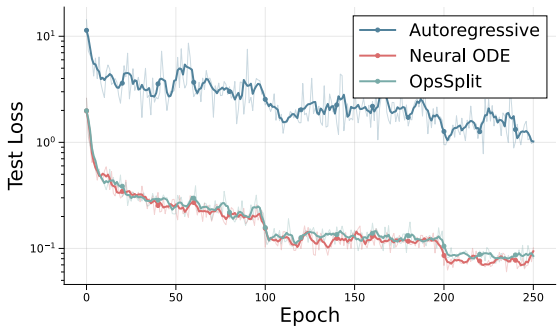
Figure 14: Incompressible Navier–Stokes: Impact of finesse of ODE solver used for temporal integration of the FNO deployed across each method for both in and out-of-distribution.

I Convergence

I.1 Pre-training: Training from Scratch

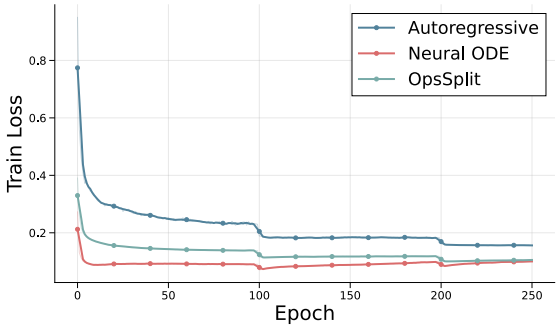


(a) Train loss across methods: FNO

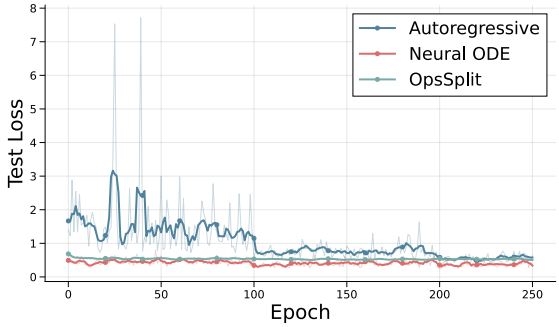


(b) Test loss across methods: FNO

Figure 15: Incompressible Navier–Stokes: Train and test loss convergences across training methods for the FNO



(a) Train loss across methods: FNO



(b) Test loss across methods: FNO

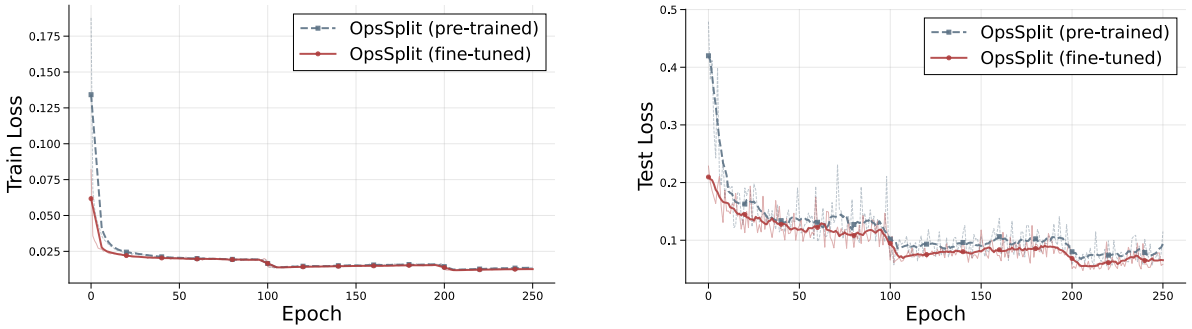
Figure 16: Compressible Navier–Stokes: Train and test loss convergences across training methods for the FNO

Figures 15 and 16 illustrate the training and test loss convergence characteristics for three distinct neural operator deployment strategies applied to fluid dynamics problems. All models are trained from scratch using random initialisation. The comparison reveals fundamental differences in how each approach learns to approximate partial differential equations. Figure 15 (Incompressible Navier–Stokes) exhibits dramatic differences in convergence behaviour across methods. The autoregressive approach (blue) maintains relatively elevated loss values throughout training, as evidenced by the logarithmic scale, indicating difficulty in learning the solution operator mapping directly. In contrast, the neural ODE method (red) achieves superior convergence by learning the dynamics operator, while OpsSplit (green) demonstrates comparable convergence characteristics to the neural ODE approach.

Figure 16 (Compressible Navier–Stokes) reveals analogous convergence patterns with quantitatively different magnitudes. The compressible formulation presents greater complexity due to coupled density-velocity-pressure dynamics, resulting in a more challenging learning problem than the incompressible case. While the neural ODE exhibits earlier convergence during training, the OpsSplit method achieves comparable final performance by the end of the optimisation process. The central insight from these convergence analyses is that both neural ODE and OpsSplit methods attain faster and more stable convergence by learning spatial dynamics explicitly and integrating predictions temporally, in contrast to the direct solution operator mapping employed by autoregressive approaches.

I.2 Fine-tuning: Transfer learning operators across PDEs

The modular structure of the OpsSplit framework enables a unique advantage: physical operators learned for one PDE system can be transferred and fine-tuned for related systems. This section investigates whether pre-trained convection operators from one fluid dynamics regime can accelerate convergence when applied to a different regime. We conduct bidirectional transfer learning experiments between the incompressible and compressible Navier–Stokes equations. For the incompressible case, we initialise the convection operator $\mathbb{NO}_{conv}$ in eq. (16) using weights pre-trained on the compressible system (eq. (21)). Conversely, for the compressible case, we initialise the convection operator in eq. (21) with weights from the incompressible formulation (eq. (16)). These fine-tuned models are compared against pre-trained models where all operators are learned from scratch using random initialisation.

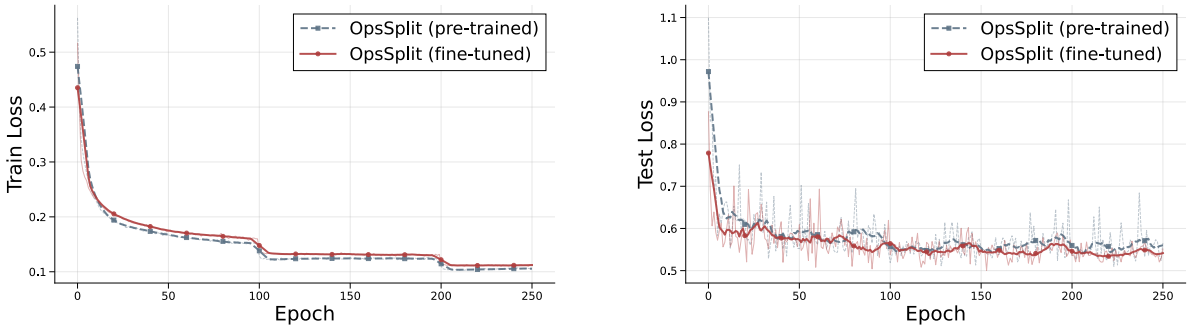


(a) Training loss convergence

(b) Test loss convergence

Figure 17: Incompressible Navier–Stokes: Comparison of training and test loss convergence for FNO-based OpsSplit models. The **pre-trained** baseline learns all operators from scratch with random initialisation. The **fine-tuned** model initialises the convection operator using weights pre-trained on the compressible Navier–Stokes equations (eq. (21)), demonstrating accelerated convergence through cross-PDE transfer learning.

Figure 17 presents convergence results for the incompressible Navier–Stokes equations. The training loss curves (fig. 17a) show minimal difference between fine-tuned and pre-trained approaches, suggesting both methods achieve similar optimisation trajectories on the training data. However, the test loss (fig. 17b) reveals a substantial advantage for transfer learning: the fine-tuned model achieves significantly lower test error and demonstrates faster convergence with identical computational resources. This indicates that the convection operator learned from compressible flow contains generalizable features that transfer effectively to incompressible flow physics.



(a) Training loss convergence

(b) Test loss convergence

Figure 18: Compressible Navier–Stokes: Comparison of training and test loss convergence for FNO-based OpsSplit models. The **pre-trained** baseline learns all operators from scratch with random initialisation. The **fine-tuned** model initialises the convection operator using weights pre-trained on the incompressible Navier–Stokes equations (eq. (16)), demonstrating that transfer learning benefits are bidirectional across fluid dynamics regimes.

Figure 18 demonstrates similar behavior for the compressible case. The fine-tuned model, initialised with the

incompressible convection operator, exhibits markedly improved test loss convergence (fig. 18b) compared to training from scratch, despite comparable training loss trajectories (fig. 18a). This bidirectional transferability confirms that the learned representations capture fundamental fluid dynamics principles that transcend specific formulations.

These results establish that neural operators trained as physical operators offer benefits beyond improved generalisation within a single PDE family. The learned operator representations can be leveraged for transfer learning across related PDE systems, enabling faster convergence and better generalisation when adapting to new physical regimes. This modularity and transferability distinguish the OpsSplit approach from monolithic neural PDE solvers, opening pathways for building reusable libraries of learned physical operators.

J Incompressible Navier–Stokes Equations

J.1 Physics

Consider the two-dimensional incompressible Navier–Stokes equations with a fixed density:

$$\nabla \cdot \mathbf{v} = 0,$$
$$\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v} = \nu \nabla^2 \mathbf{v} - \nabla P,$$

where under constant density the pressure is given via the pressure Poisson formulation:

$$\nabla^2 P = -\nabla \cdot [(\mathbf{v} \cdot \nabla)\mathbf{v}] \tag{34}$$
$$\tag{35}$$

with initial conditions:

$$u(x, y, t = 0) = -\sin(2\pi\alpha y) \quad y \in [-1, 1], \tag{36}$$
$$v(x, y, t = 0) = -\sin(4\pi\beta x) \quad x \in [-1, 1], \tag{37}$$

where u defines the x-component of velocity, v defines the y-component of velocity. The Navier–Stokes equations solve the flow of an incompressible fluid with a kinematic viscosity ν. The system is bounded with periodic boundary conditions within the domain. The dataset is built by performing a Latin hypercube scan across the defined domain for the parameters α, β, which parametrise the initial velocity fields for each simulation. We generate 500 simulation points, each with its initial condition and use them for training. The solver is built using a spectral method outlined in Philip Mocz's code.

Each data point, as in each simulation, is generated with a different initial condition as described above. The parameters of the initial conditions are sampled from within the domain as given in table 7. Each simulation is run up until wallclock time reaches 0.5 $\Delta t = 0.001$. The spatial domain is uniformly discretised into 400 spatial units in the x and y axes. The temporal domain is subsampled to factor in every 10^{th} time instance, and the spatial domain is downsampled to factor every 4^{th} time instance, leading to a 100×100 grid for the neural PDE. Parameterisation of the initial conditions and the kinematic viscosity used for both training and testing can be found in table 7.

Table 7: Parameterisations of the 2D incompressible Navier–Stokes equations utilised for training and OOD testing

Parameter	Training	OOD Testing	Type
Velocity x-axis (u_0)	$[0.5, 1.0]$	$[0.1, 0.5]$	Continuous
Velocity y-axis (v_0)	$[0.5, 1.0]$	$[0.1, 0.5]$	Continuous
viscosity ν	0.001	0.01	Discrete

J.2 Model Details

We evaluated five neural operator architectures within each deployment method: Fourier Neural Operator (FNO) [Li et al., 2021], U-Net [Ronneberger et al., 2015, Gupta and Brandstetter, 2023], Vision Transformer (ViT) [Dosovitskiy et al., 2021, Herde et al., 2024], U-shaped Neural Operator (UNO) [Rahman et al., 2023], and Convolutional Neural Operator (CNO) [Bartolucci et al., 2023]. All models processed 2-channel velocity field inputs and outputs with configurations detailed in Table below. For each model, the hyperparameters were chosen inspired from the literature and constructed to maximise the GPU utilisation within a single H100 GPU.

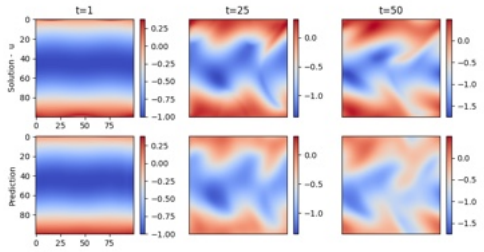
Model	Configuration Details - AR, NODE, OpsSplit
FNO	in_channels: 2 out_channels: 2 modes: 32 width: 64 n_layers: 6 activation_func: GeLU
U-Net	in_channels: 2 out_channels: 2 initial_width: 64 activation_func: Tanh
ViT	patch_size: 4 embed_dim: 512 in_channels: 2 out_channels: 2 time_channels: 1 depth: 12 num_heads: 10 activation_func: GeLU
CNO	Nx: 100 N_layers: 4 N_res: 4 N_res_neck: 2 in_channels: 2 out_channels: 2 channel_multiplier: 16 activation_func: LReLU
UNO	arch: UNO in_channels: 2 out_channels: 2 width: 64 projection_channels: 256 n_layers: 5 uno_out_channels: [32,64,64,64,32] uno_n_modes: [[16,16],[8,8],[8,8],[8,8],[16,16]] domain_padding: 0.2 norm: group_norm activation_func: GeLU

Table 8: Model configuration details of neural operators used for modelling the incompressible Navier–Stokes equations. All three methods - AR, NODE, and OpsSplit utilised NO models with the same configuration as mentioned above.

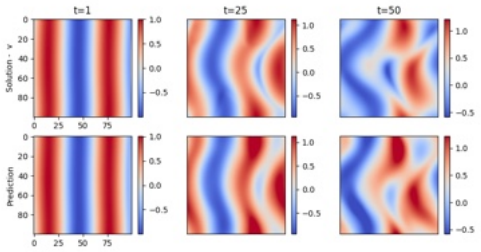
J.3 Performance Plots

In this section, we showcase figures to provide a qualitative comparison of model predictions across different deployment methods and neural operator architectures. Each visualisation displays the ground truth (top row) and model predictions (bottom row) at three representative time instances: t=1 (early-stage dynamics within training temporal resolution), t=25 (mid-simulation behaviour testing model stability), and t=50 (long-term temporal extrapolation). Velocity field visualisations are represented in the physical space within the Cartesian domain. These visualisations enable assessment of spatial accuracy (how well the model captures field patterns and structures), temporal stability (whether predictions maintain physical consistency over time), error accumulation (how prediction errors grow during autoregressive rollout or temporal extrapolation), and method comparison (relative performance of Autoregressive, Neural ODE, and OpsSplit approaches across different architectures). For quantitative metrics corresponding to these visualisations, refer to table 1 in the main text.

J.3.1 FNO

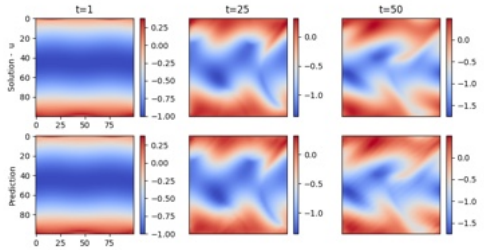


(a) Horizontal velocity - Autoregressive FNO

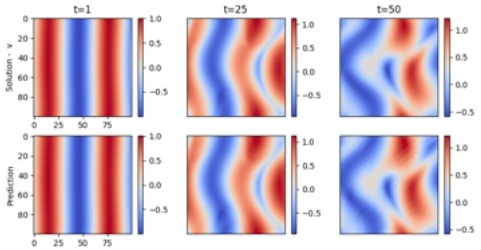


(b) Vertical velocity - Autoregressive FNO

Figure 19: Incompressible Navier–Stokes: Model prediction within test distribution for Autoregressive FNO

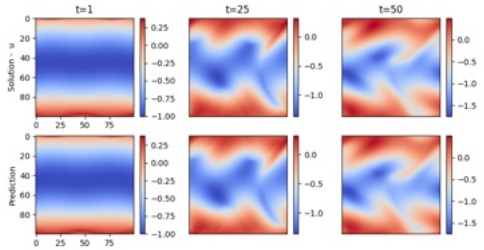


(a) Horizontal velocity - NODE FNO

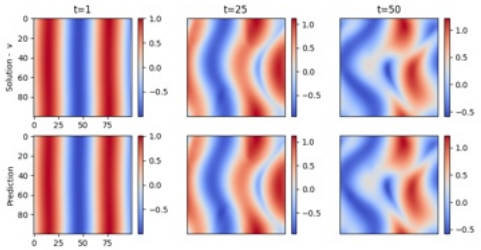


(b) Vertical velocity - NODE FNO

Figure 20: Incompressible Navier–Stokes: Model prediction within test distribution for Neural-ODE FNO



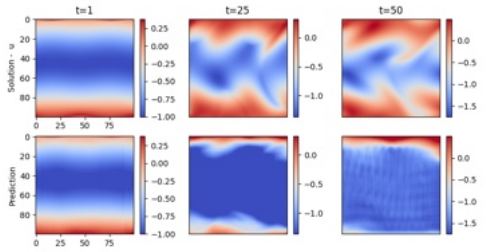
(a) Horizontal velocity - OpsSplit FNO



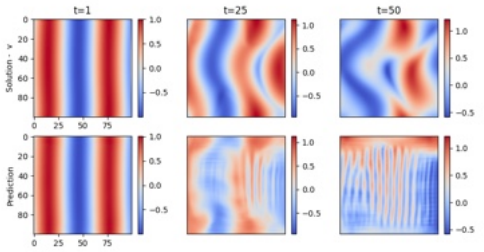
(b) Vertical velocity - OpsSplit FNO

Figure 21: Incompressible Navier–Stokes: Model prediction within test distribution for OpsSplit FNO

J.3.2 U-Net

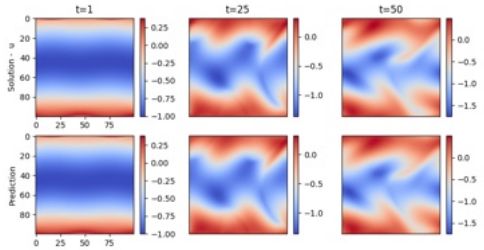


(a) Horizontal velocity - Autoregressive U-Net

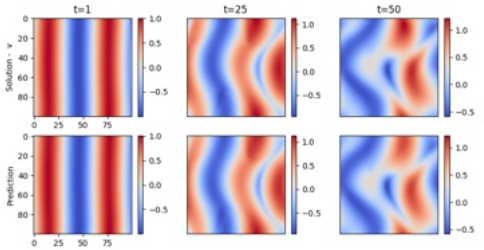


(b) Vertical velocity - Autoregressive U-Net

Figure 22: Incompressible Navier–Stokes: Model prediction within test distribution for Autoregressive U-Net

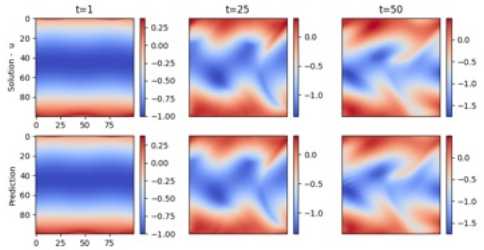


(a) Horizontal velocity - NODE U-Net

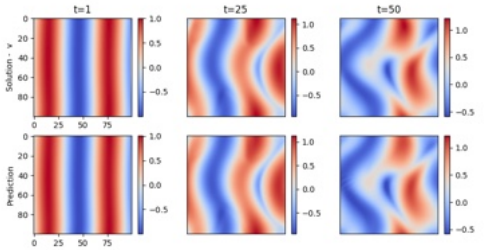


(b) Vertical velocity - NODE U-Net

Figure 23: Incompressible Navier–Stokes: Model prediction within test distribution for Neural-ODE U-Net



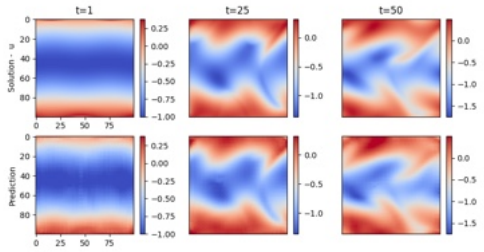
(a) Horizontal velocity - OpsSplit U-Net



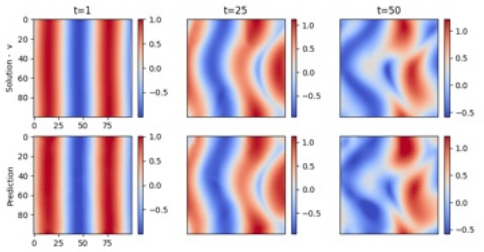
(b) Vertical velocity - OpsSplit U-Net

Figure 24: Incompressible Navier–Stokes: Model prediction within test distribution for OpsSplit U-Net

J.3.3 ViT

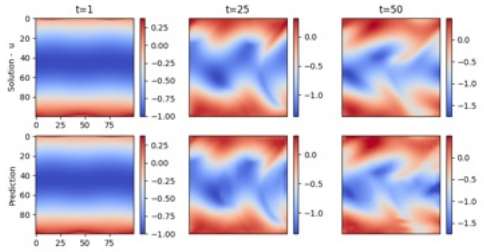


(a) Horizontal velocity - Autoregressive ViT

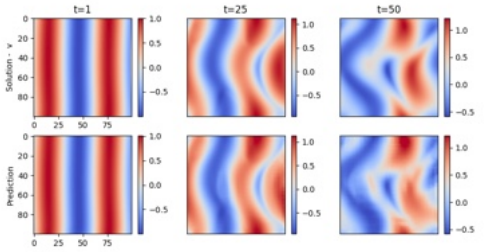


(b) Vertical velocity - Autoregressive ViT

Figure 25: Incompressible Navier–Stokes: Model prediction within test distribution for Autoregressive ViT

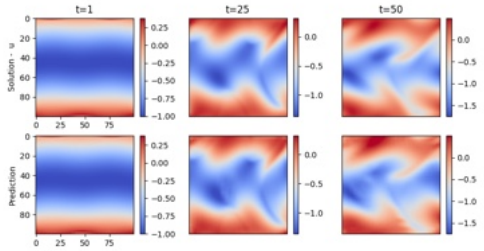


(a) Horizontal velocity - NODE ViT

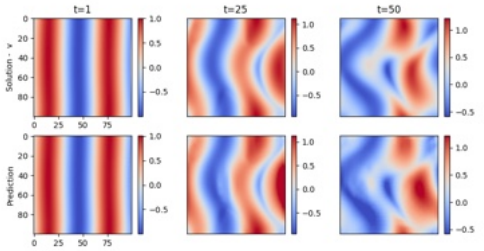


(b) Vertical velocity - NODE ViT

Figure 26: Incompressible Navier–Stokes: Model prediction within test distribution for Neural-ODE ViT



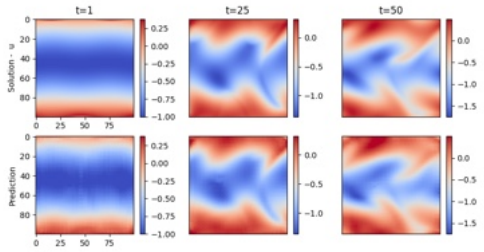
(a) Horizontal velocity - OpsSplit ViT



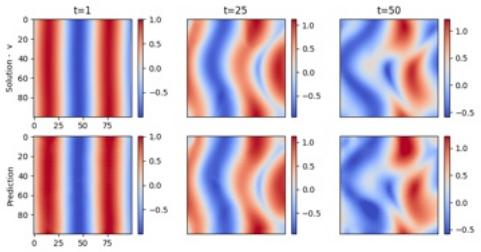
(b) Vertical velocity - OpsSplit ViT

Figure 27: Incompressible Navier–Stokes: Model prediction within test distribution for OpsSplit ViT

J.3.4 CNO

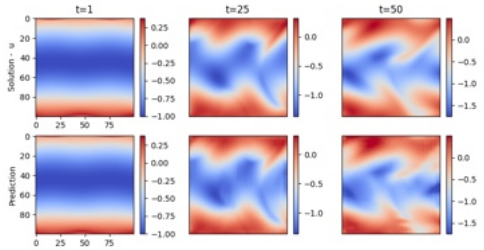


(a) Horizontal velocity - Autoregressive CNO

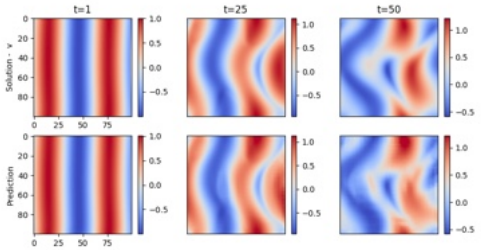


(b) Vertical velocity - Autoregressive CNO

Figure 28: Incompressible Navier–Stokes: Model prediction within test distribution for Autoregressive CNO

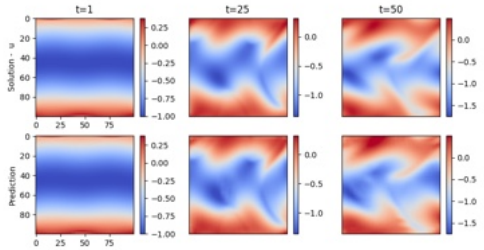


(a) Horizontal velocity - NODE CNO

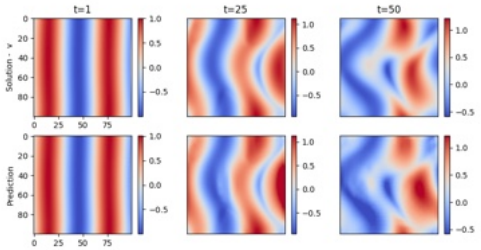


(b) Vertical velocity - NODE CNO

Figure 29: Incompressible Navier–Stokes: Model prediction within test distribution for Neural-ODE CNO



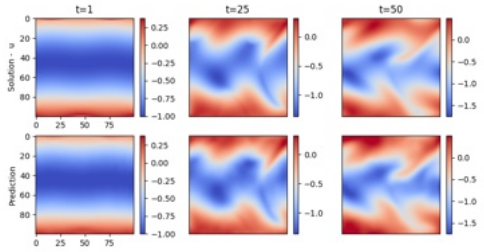
(a) Horizontal velocity - OpsSplit CNO



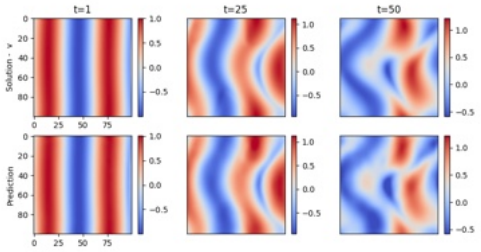
(b) Vertical velocity - OpsSplit CNO

Figure 30: Incompressible Navier–Stokes: Model prediction within test distribution for OpsSplit CNO

J.3.5 UNO

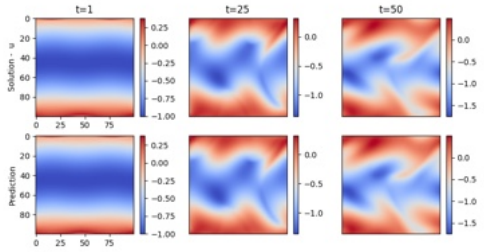


(a) Horizontal velocity - Autoregressive UNO

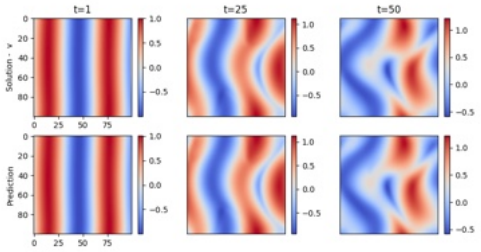


(b) Vertical velocity - Autoregressive UNO

Figure 31: Incompressible Navier–Stokes: Model prediction within test distribution for Autoregressive UNO

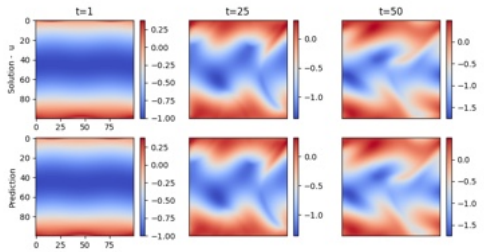


(a) Horizontal velocity - NODE UNO

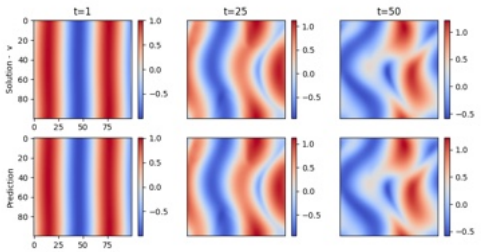


(b) Vertical velocity - NODE UNO

Figure 32: Incompressible Navier–Stokes: Model prediction within test distribution for Neural-ODE UNO



(a) Horizontal velocity - OpsSplit UNO



(b) Vertical velocity - OpsSplit UNO

Figure 33: Incompressible Navier–Stokes: Model prediction within test distribution for OpsSplit UNO

K Compressible Navier–Stokes Equations

K.1 Physics

Consider the two-dimensional compressible Navier–Stokes equations under adiabatic and inviscid flow:

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot (\rho \mathbf{v}),$$

$$\frac{\partial \mathbf{v}}{\partial t} = -(\mathbf{v} \cdot \nabla)\mathbf{v} - \frac{1}{\rho}\nabla P,$$

$$\frac{\partial P}{\partial t} = -\mathbf{v} \cdot \nabla P - \gamma P(\nabla \cdot \mathbf{v}),$$

where ρ determines the density, u defines the x-component of velocity, v defines the y-component of velocity and P determines the pressure of the fluid. The Navier–Stokes equations solve the flow of a compressible fluid given by its specific heat ratio of γ. The system defines the flow without viscosity, bounded with periodic boundary conditions within the domain, modelling two opposite moving streams under perturbation. The dataset is built by performing a Latin hypercube scan across the defined domain for the parameters α, β, which parametrise the initial velocity and pressure fields for each simulation as given in eqs. (38) to (41). We generate 500 simulation points, each with its initial condition and use them for training. The solver is built using a finite-volume method to solve for the Kelvin-Helmholtz instability as outlined in Philip Mocz's code.

$$\rho = \begin{cases} 2 & \text{if } |Y - 0.5| < 0.25 \\ 1 & \text{otherwise} \end{cases} \tag{38}$$

$$v_x = \begin{cases} 0.5 & \text{if } |Y - 0.5| < 0.25 \\ -0.5 & \text{otherwise} \end{cases} \tag{39}$$

$$v_y = \alpha \sin(4\pi X) \left[\exp\left(-\frac{(Y - 0.25)^2}{2\sigma^2} \right) + \exp\left(-\frac{(Y - 0.75)^2}{2\sigma^2} \right) \right] ; \sigma = \frac{0.05}{\sqrt{2}} \tag{40}$$

$$P = \beta \tag{41}$$

Each data point, as in each simulation, is generated with a different initial condition as described above. The parameters of the initial conditions are sampled from within the domain as given in table 9. Each simulation is run up until wall-clock time reaches 2.0 $\Delta t = 0.0005$. The spatial domain is uniformly discretised into 128 spatial units in the x and y axes. The temporal domain is subsampled to factor in every 20^{th} time instance, and the spatial domain is kept as is with a 128×128 grid for the neural PDE. Parameterisation of the initial conditions and the gas ratio used for both training and testing can be found in table 9.

Table 9: Parameterisation of the 2D Euler Fluid equations utilised for training and OOD testing

Parameter	Training	OOD Testing	Type
α	$[0.1, 0.5]$	$[0.5, 1.0]$	Continuous
β	$[1.0, 5.0]$	$[5.0, 10.0]$	Continuous
γ	$\frac{5}{3}$	$\frac{2}{3}$	Discrete

K.2 Model Details

We evaluated four neural operator architectures within each deployment method: Fourier Neural Operator (FNO) [Li et al., 2021], U-Net [Ronneberger et al., 2015, Gupta and Brandstetter, 2023], Vision Transformer (ViT) [Dosovitskiy et al., 2021, Herde et al., 2024] and U-shaped Neural Operator (UNO) [Rahman et al., 2023]. All models processed 4 variables, the velocity vector field and the scalar fields associated with density and pressure. For each model, the hyperparameters were chosen inspired from the literature and constructed to maximise the GPU utilisation within a single H100 GPU as given below:

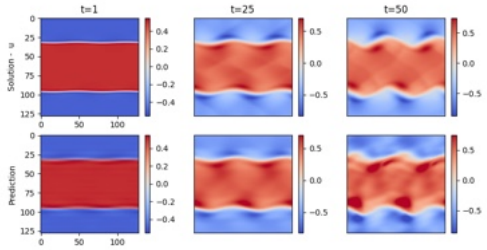
Model	Configuration Details - AR, NODE	OpsSplit
FNO	in_channels: 4 out_channels: 4 modes: 32 width: 64 n_layers: 6 activation_func: GeLU	in_channels: 2 (conv), 3 (div) out_channels: 2 (conv), 1 (div) modes: 32 width: 64 n_layers: 3 activation_func: GeLU
U-Net	in_channels: 4 out_channels: 4 initial_width: 64 activation_func: Tanh	in_channels: 2 (conv), 3 (div) out_channels: 2 (conv), 1 (div) initial_width: 32 activation_func: Tanh
ViT	patch_size: 4 embed_dim: 512 in_channels: 4 out_channels: 4 time_channels: 1 depth: 12 num_heads: 10 activation_func: GeLU	patch_size: 4 embed_dim: 256 in_channels: 2 (conv), 3 (div) out_channels: 2 (conv), 1 (div) time_channels: 1 depth: 8 num_heads: 10 activation_func: GeLU
UNO	in_channels: 4 out_channels: 4 width: 64 projection_channels: 256 n_layers: 5 uno_out_channels: [32,64,64,64,32] uno_n_modes: [[16,16],[8,8],[8,8],[8,8],[16,16]] domain_padding: 0.2 norm: group_norm activation_func: GeLU	in_channels: 2 (conv), 3 (div) out_channels: 2 (conv), 1 (div) width: 32 projection_channels: 256 n_layers: 5 uno_out_channels: [32,64,64,64,32] uno_n_modes: [[16,16],[8,8],[8,8],[8,8],[16,16]] domain_padding: 0.2 norm: group_norm activation_func: GeLU

Table 10: Model configuration details of neural operators used for modelling the compressible Navier–Stokes equations. Both AR and NODE-based models have the same configuration, whereas the two models within OpsSplit have slightly different setups to ensure the same parameterisation levels across all methods.

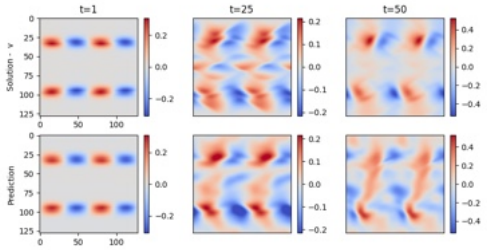
K.3 Performance Plots

In this section, we showcase figures to provide a qualitative comparison of model predictions across different deployment methods and neural operator architectures. Each visualisation displays the ground truth (top row) and model predictions (bottom row) at three representative time instances: t=1 (early-stage dynamics within training temporal resolution), t=25 (mid-simulation behaviour testing model stability), and t=50 (long-term temporal extrapolation). Velocity and scalar field visualisations are represented in the physical space within the Cartesian domain. These visualisations enable assessment of spatial accuracy (how well the model captures field patterns and structures), temporal stability (whether predictions maintain physical consistency over time), error accumulation (how prediction errors grow during autoregressive rollout or temporal extrapolation), and method comparison (relative performance of Autoregressive, Neural ODE, and OpsSplit approaches across different architectures). For quantitative metrics corresponding to these visualisations, refer to table 2 in the main text.

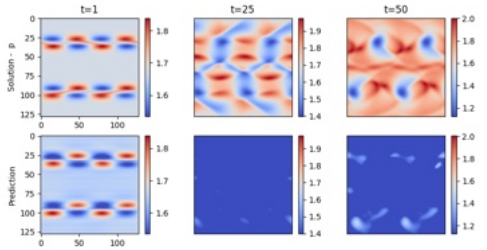
K.3.1 FNO



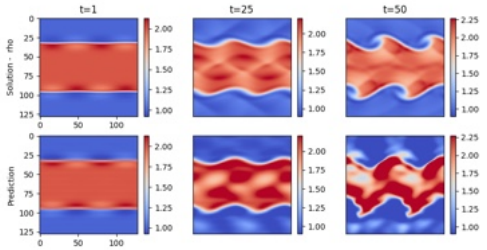
(a) Horizontal velocity - Autoregressive FNO



(b) Vertical velocity - Autoregressive FNO

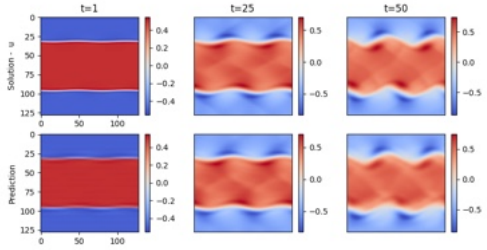


(c) Pressure - Autoregressive FNO

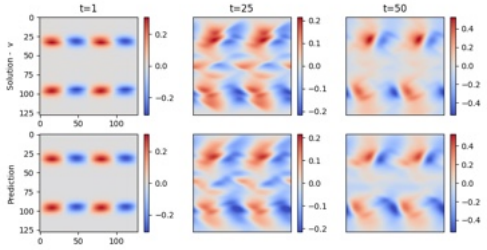


(d) Density - Autoregressive FNO

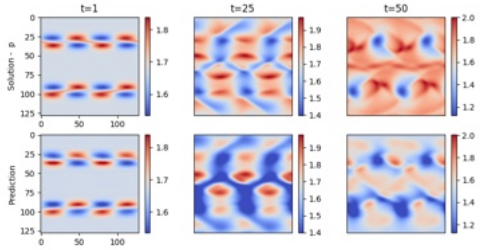
Figure 34: Compressible Navier–Stokes: Model prediction within test distribution for Autoregressive FNO



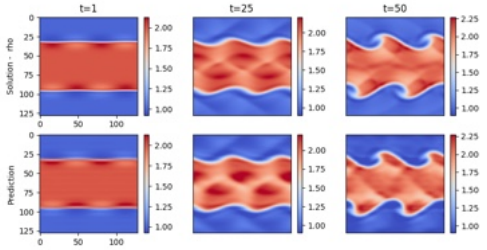
(a) Horizontal velocity - NODE FNO



(b) Vertical velocity - NODE FNO

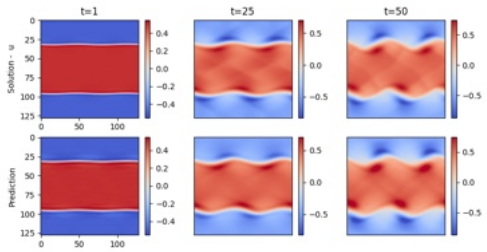


(c) Pressure - NODE FNO

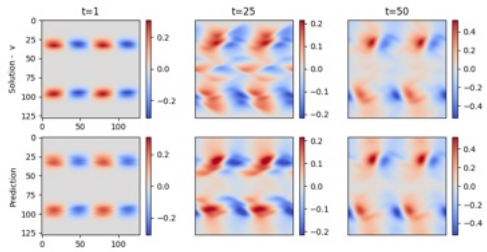


(d) Density - NODE FNO

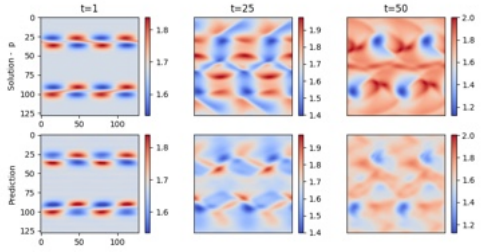
Figure 35: Compressible Navier–Stokes: Model prediction within test distribution for Neural-ODE FNO



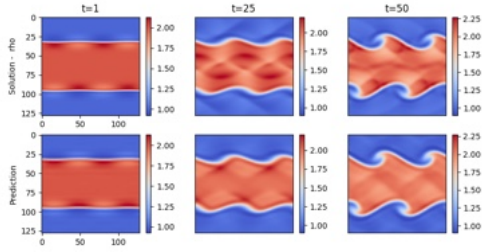
(a) Horizontal velocity - OpsSplit FNO



(b) Vertical velocity - OpsSplit FNO



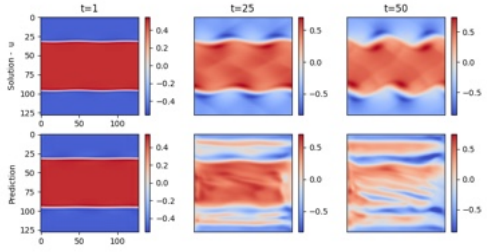
(c) Pressure - OpsSplit FNO



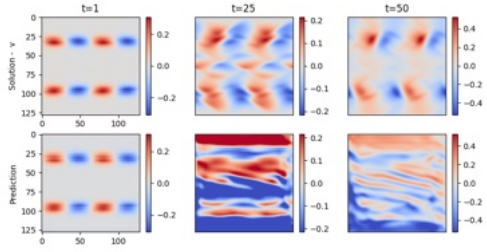
(d) Density - OpsSplit FNO

Figure 36: Compressible Navier–Stokes: Model prediction within test distribution for OpsSplit FNO

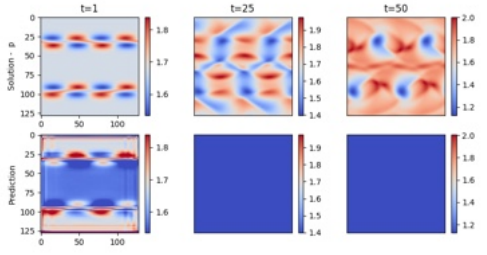
K.3.2 U-Net



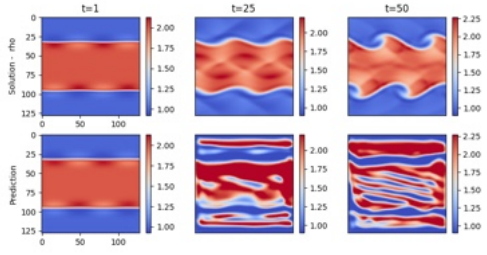
(a) Horizontal velocity - Autoregressive U-Net



(b) Vertical velocity - Autoregressive U-Net

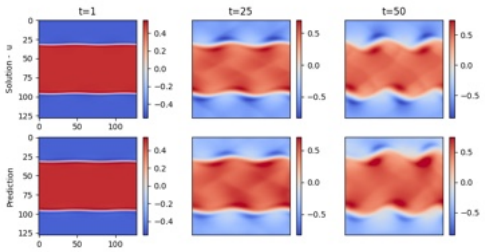


(c) Pressure - Autoregressive U-Net

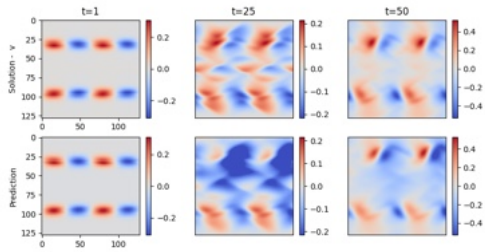


(d) Density - Autoregressive U-Net

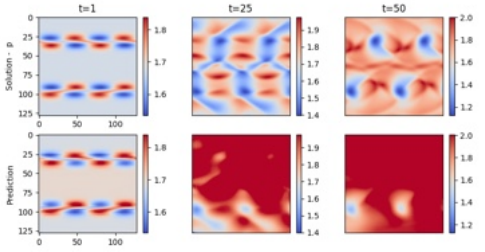
Figure 37: Compressible Navier–Stokes: Model prediction within test distribution for Autoregressive U-Net



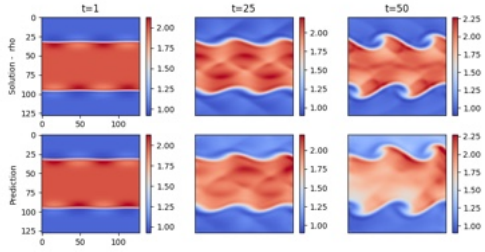
(a) Horizontal velocity - NODE U-Net



(b) Vertical velocity - NODE U-Net

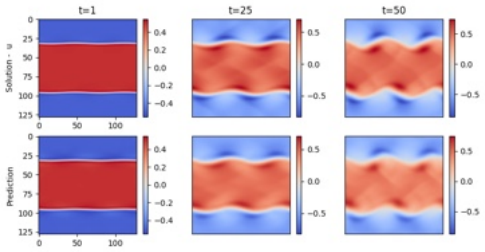


(c) Pressure - NODE U-Net

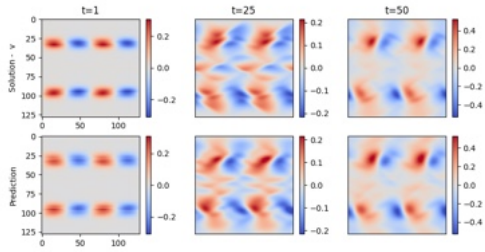


(d) Density - NODE U-Net

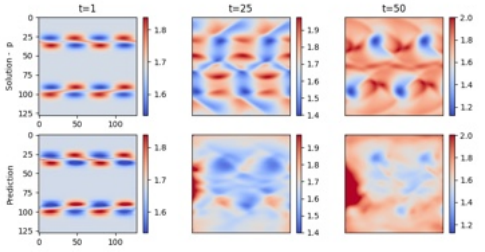
Figure 38: Compressible Navier–Stokes: Model prediction within test distribution for Neural-ODE U-Net



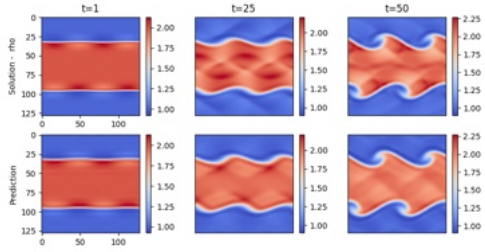
(a) Horizontal velocity - OpsSplit U-Net



(b) Vertical velocity - OpsSplit U-Net



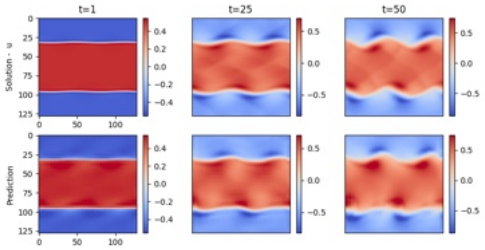
(c) Pressure - OpsSplit U-Net



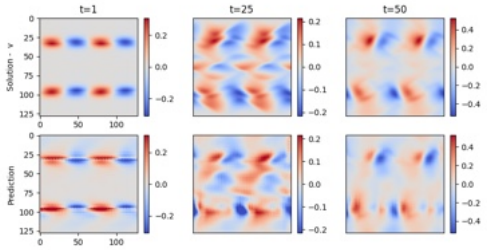
(d) Density - OpsSplit U-Net

Figure 39: Compressible Navier–Stokes: Model prediction within test distribution for OpsSplit U-Net

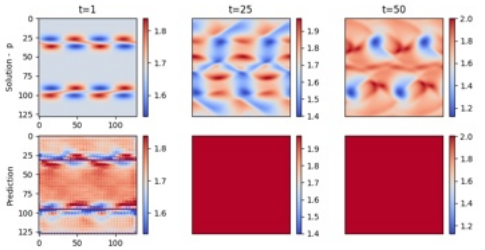
K.3.3 ViT



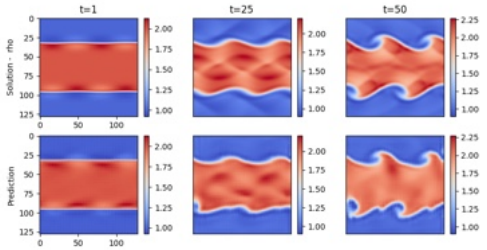
(a) Horizontal velocity - Autoregressive ViT



(b) Vertical velocity - Autoregressive ViT

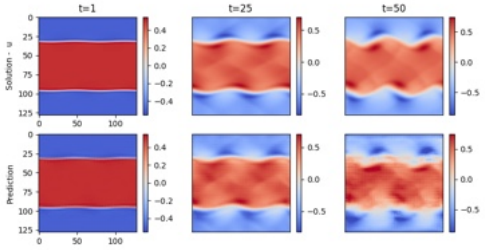


(c) Pressure - Autoregressive ViT

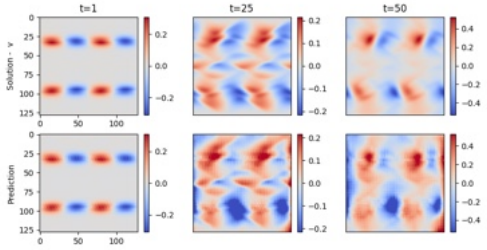


(d) Density - Autoregressive ViT

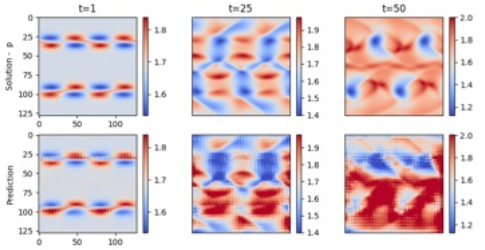
Figure 40: Compressible Navier–Stokes: Model prediction within test distribution for Autoregressive ViT



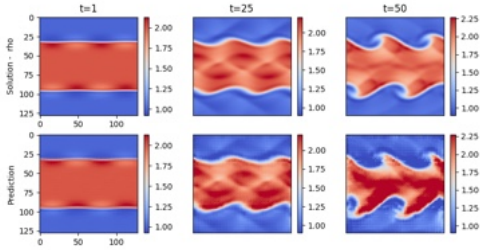
(a) Horizontal velocity - NODE ViT



(b) Vertical velocity - NODE ViT

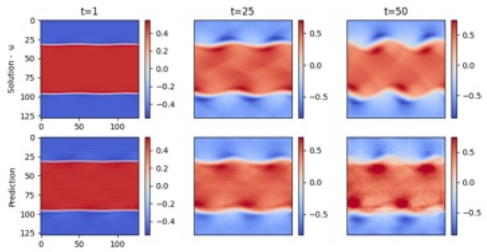


(c) Pressure - NODE ViT

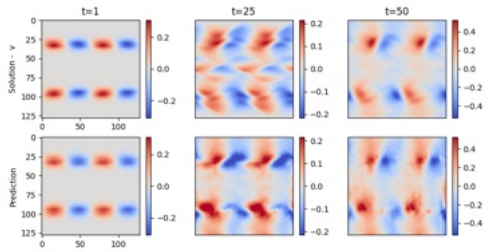


(d) Density - NODE ViT

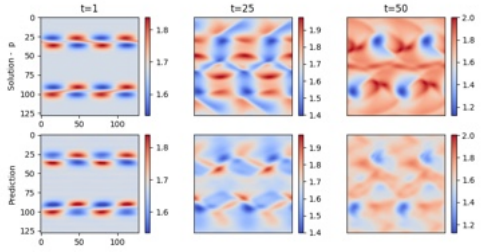
Figure 41: Compressible Navier–Stokes: Model prediction within test distribution for Neural-ODE ViT



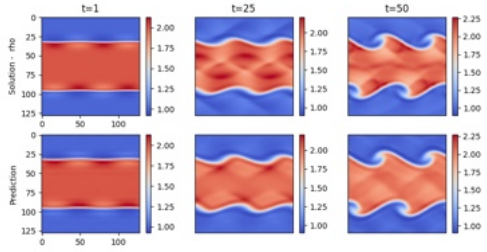
(a) Horizontal velocity - OpsSplit ViT



(b) Vertical velocity - OpsSplit ViT



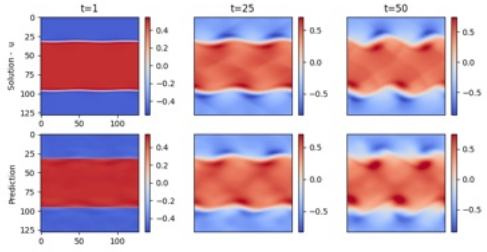
(c) Pressure - OpsSplit ViT



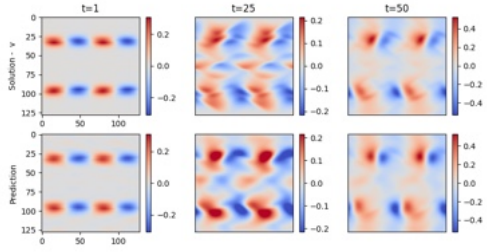
(d) Density - OpsSplit ViT

Figure 42: Compressible Navier–Stokes: Model prediction within test distribution for OpsSplit ViT

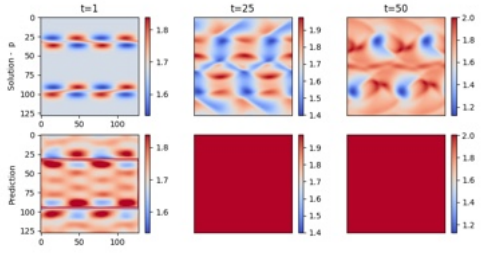
K.3.4 UNO



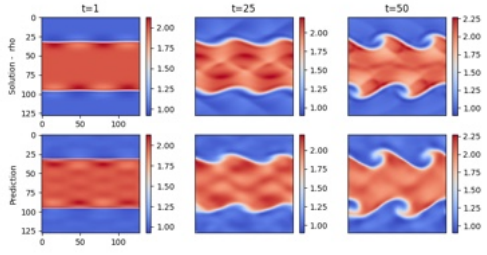
(a) Horizontal velocity - Autoregressive UNO



(b) Vertical velocity - Autoregressive UNO

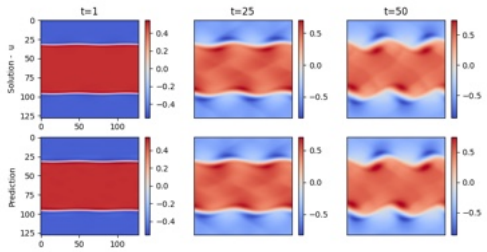


(c) Pressure - Autoregressive UNO

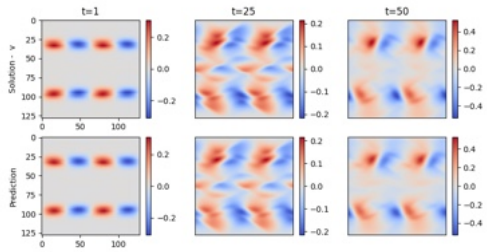


(d) Density - Autoregressive UNO

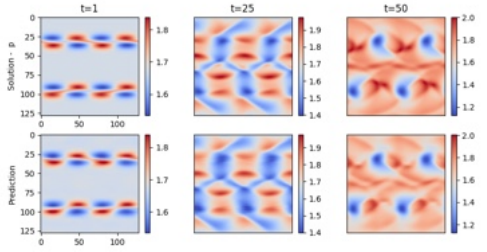
Figure 43: Compressible Navier–Stokes: Model prediction within test distribution for Autoregressive UNO



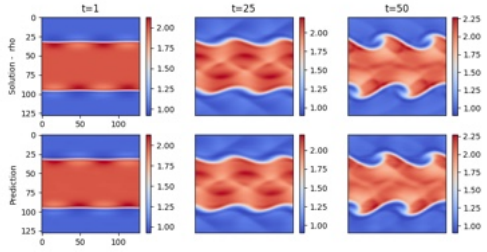
(a) Horizontal velocity - NODE UNO



(b) Vertical velocity - NODE UNO

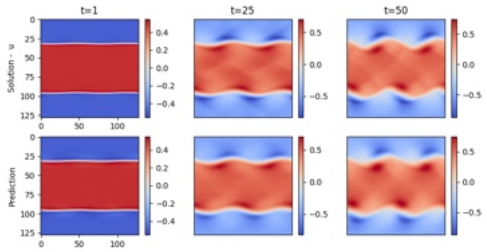


(c) Pressure - NODE UNO

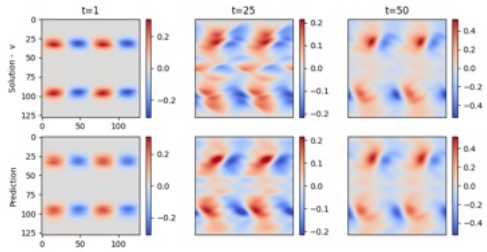


(d) Density - NODE UNO

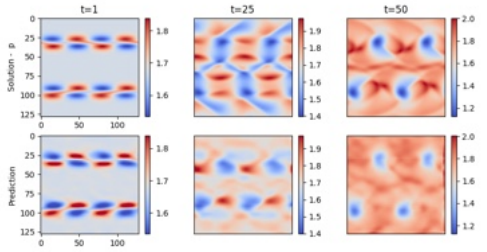
Figure 44: Compressible Navier–Stokes: Model prediction within test distribution for Neural-ODE UNO



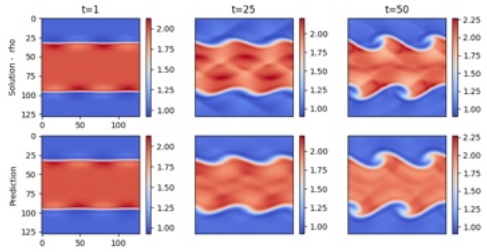
(a) Horizontal velocity - OpsSplit UNO



(b) Vertical velocity - OpsSplit UNO



(c) Pressure - OpsSplit UNO



(d) Density - OpsSplit UNO

Figure 45: Compressible Navier–Stokes: Model prediction within test distribution for OpsSplit UNO