POSEIDON: Efficient Foundation Models for PDEs

Maximilian Herde[1,*] **Bogdan Raonić**[1,2,*] **Tobias Rohner**[1] **Roger Käppeli**[1]

Roberto Molinaro[1] **Emmanuel de Bézenac**[1] **Siddhartha Mishra**[1,2]

[1]Seminar for Applied Mathematics, ETH Zurich, Switzerland
[2]ETH AI Center, Zurich, Switzerland
Correspondence to herdem@ethz.ch

Abstract

We introduce POSEIDON, a foundation model for learning the solution operators of PDEs. It is based on a multiscale operator transformer, with time-conditioned layer norms that enable continuous-in-time evaluations. A novel training strategy leveraging the semi-group property of time-dependent PDEs to allow for significant scaling-up of the training data is also proposed. POSEIDON is pretrained on a diverse, large scale dataset for the governing equations of fluid dynamics. It is then evaluated on a suite of 15 challenging downstream tasks that include a wide variety of PDE types and operators. We show that POSEIDON exhibits excellent performance across the board by outperforming baselines significantly, both in terms of sample efficiency and accuracy. POSEIDON also generalizes very well to new physics that is not seen during pretraining. Moreover, POSEIDON scales with respect to model and data size, both for pretraining and for downstream tasks. Taken together, our results showcase the surprising ability of POSEIDON to learn effective representations from a very small set of PDEs during pretraining in order to generalize well to unseen and unrelated PDEs downstream, demonstrating its potential as an effective, general purpose PDE foundation model. Finally, the POSEIDON model as well as underlying pretraining and downstream datasets are open sourced, with code being available at https://github.com/camlab-ethz/poseidon and pretrained models and datasets at https://huggingface.co/camlab-ethz.

1 Introduction

Partial Differential Equations (PDEs) [15] are referred to as the *language* of physics as they mathematically model a very wide variety of physical phenomena across a vast range of spatio-temporal scales. *Numerical methods* such as finite difference, finite element, spectral methods etc. [59] are commonly used to approximate or *simulate* PDEs. However, their (prohibitive) computational cost, particularly for the so-called many-query problems [58], has prompted the design of various *data-driven* machine learning (ML) methods for simulating PDEs, [24, 51] and references therein. Among them, *operator learning* algorithms have gained increasing traction in recent years.

These methods aim to learn the underlying PDE solution operator, which maps function space inputs (initial and boundary conditions, coefficients, sources) to the PDE solution. They include algorithms which approximate a *discretization*, on a fixed grid, of the underlying solution operator. These can be based on convolutions [75, 18], graph neural networks [8, 56, 65] or transformers [12, 57, 26, 20, 35]. Other operator learning algorithms are *neural operators* which can directly

[*] Equal contribution

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

process function space inputs and outputs, possibly sampled on multiple grid resolutions [27, 3]. These include DeepONets [13, 42], Fourier Neural Operator [33], SFNO [7], Geo-FNO [32], Low-rank NO [34] and Convolutional Neural Operator [60], among many others.

However, existing operator learning methods are not *sample efficient* as they can require a very large number of training examples to learn the target solution operator with desired accuracy (see Figure 1 or Figure 3 of [60]). This impedes their widespread use as *task-specific* training data is very expensive to generate either with numerical simulations or measurements of the underlying physical system.

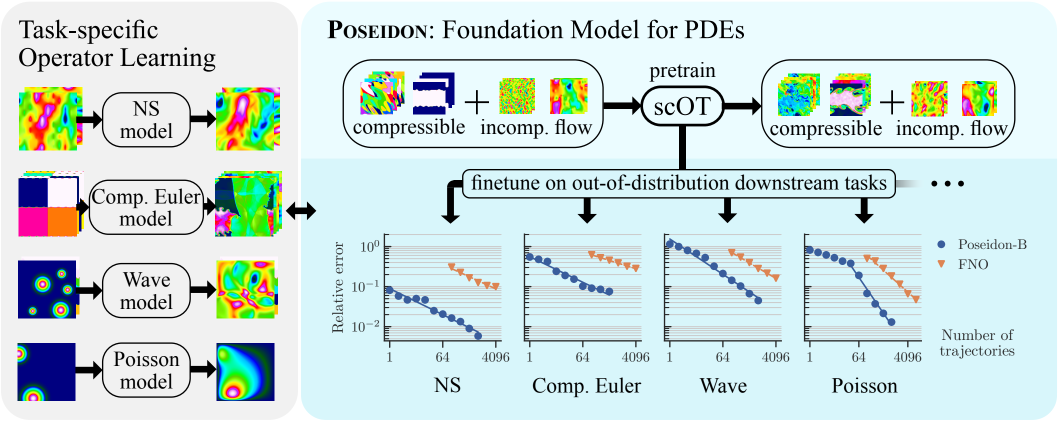


Figure 1: As opposed to PDE-specific operator learning, our pretrained model POSEIDON is up to multiple orders of magnitude more sample efficient than a task-specific neural operator while also being able to transfer to unseen physics during finetuning.

How can the number of training samples for PDE learning be significantly reduced? In this context, can we learn from language modeling and computer vision where a similar question often arises and the current paradigm is to build *foundation models* [6]. These *generalist* models are *pretrained*, at-scale, on large datasets drawn from a diverse set of data distributions. They leverage the intrinsic ability of neural networks to learn *effective representations* from pretraining and are then successfully deployed on a variety of *downstream* tasks by *finetuning* them on a few task-specific samples. Examples of such models include highly successful large language models [10, 72], large multi-modal models[17, 52] and foundation models for robotics [9], chemistry [4], biology [63], medicine [64] and climate [54].

Despite very recent preliminary attempts [67, 74, 1, 49, 19, 68, 66], the challenge of designing such foundation models for PDEs is *formidable*, given the sheer variety of PDEs (linear and nonlinear, steady and evolutionary, elliptic, parabolic, hyperbolic and mixed etc.), the immense diversity of data distributions, wide range of underlying spatio-temporal scales and the paucity of publicly available high-quality datasets. In particular, the very feasibility of designing PDE foundation models rests on the fundamental and unanswered science question of *why pretraining a model on a (very) small set of PDEs and underlying data-distributions can allow it to learn effective representations and generalize to unseen and unrelated PDEs and data-distributions via finetuning?*

The investigation of this open question motivates us here to present the POSEIDON family of PDE foundation models. POSEIDON, see Figures 1 and 2, is based on i) scalable Operator Transformer or scOT, a *multiscale vision transformer* with (shifted) windowed or Swin attention [38, 37], adapted for operator learning, ii) a novel all2all training strategy for efficiently leveraging *trajectories* of solutions of time-dependent PDEs to scale up the volume of training data and iii) an open source large-scale pretraining dataset, containing a set of novel solution operators of the compressible Euler and incompressible Navier-Stokes equations of fluid dynamics. We evaluate POSEIDON on a challenging suite of 15 downstream tasks, comprising of well-established benchmarks in computational physics that encompass linear and nonlinear, time-dependent and independent and elliptic, parabolic, hyperbolic and mixed type PDEs. All of these tasks are *out-of-distribution* with respect to the pretraining data. Moreover, nine out of the 15 tasks even involve PDEs (and underlying physical processes) which are not encountered during pretraining.

Through extensive experiments, we find that i) POSEIDON shows impressive performance across the board and outperforms baselines on the downstream tasks, with significant gains in accuracy and order of magnitude gains in sample efficiency. For instance, on an average (median) over the downstream tasks, POSEIDON requires a mere 20 samples to attain the same error level as the widely-used FNO does with 1024 samples. ii) These gains in accuracy and sample efficiency are also displayed on tasks which involve PDEs not encountered during pretraining, allowing us to conclude that POSEIDON can *generalize to unseen and a priori unrelated physical processes and phenomena* with a few task-specific training examples and iii) POSEIDON scales with model and dataset size, both for the pretraining as well as for downstream tasks and iv) through case studies, we elucidate possible mechanisms via which POSEIDON is able to learn *effective representations* during pretraining, which are then leveraged to generalize to unrelated PDEs downstream. Taken together, these results provide the first positive answers to the afore-mentioned fundamental question of the very feasibility of PDE foundation models and pave the way for the further development and deployment of POSEIDON as an *efficient general purpose PDE foundation model.* Finally, we also open source the POSEIDON model and the entire pretraining and downstream task datasets within the PDEGYM database.

2 Approach

Problem Formulation. We denote a generic time-dependent PDE as,

$$\begin{aligned} &\partial_t u(x,t) + \mathcal{L}\left(u, \nabla_x u, \nabla_x^2 u, \ldots\right) = 0, \quad \forall x \in D \subset \mathbb{R}^d, t \in (0,T), \\ &\mathcal{B}(u) = 0, \quad \forall (x,t) \in \partial D \times (0,T), \quad u(0,x) = a(x), \quad x \in D \end{aligned} \tag{1}$$

Here, with a function space $\mathcal{X} \subset L^p(D;\mathbb{R}^n)$ for some $1 \leq p < \infty$, $u \in C([0,T];\mathcal{X})$ is the solution of (1), $a \in \mathcal{X}$ the initial datum and $\mathcal{L}, \mathcal{B}$ are the underlying differential and boundary operators, respectively. Note that (1) accommodates both PDEs with high-order time-derivatives as well as PDEs with (time-independent) coefficients and sources by including the underlying functions within the solution vector and augmenting $\mathcal{L}$ accordingly (see **SM** B.2 for examples).

Even *time-independent* PDEs can be recovered from (1) by taking the *long-time limit*, i.e., $\lim_{t\to\infty} u = \overline{u}$, which will be the solution of the (generic) time-independent PDE,

$$\mathcal{L}\left(\overline{u}(x), \nabla_x \overline{u}, \nabla_x^2 \overline{u}, \ldots\right) = 0, \quad \forall x \in D, \quad \mathcal{B}(\overline{u}) = 0, \quad \forall x \in \partial D. \tag{2}$$

Solutions of the PDE (1) are given in terms of the underlying *solution operator* $\mathcal{S} : [0,T] \times \mathcal{X} \mapsto \mathcal{X}$ such that $u(t) = \mathcal{S}(t,a)$ is the solution of (1) at any time $t \in (0,T)$. Given a data distribution $\mu \in \mathrm{Prob}(\mathcal{X})$, the *underlying operator learning task (OLT)* is,

OLT: *Given any initial datum $a \sim \mu$, find an approximation $\mathcal{S}^* \approx \mathcal{S}$ to the solution operator $\mathcal{S}$ of* (1)*, in order to generate the entire solution trajectory $\{\mathcal{S}^*(t,a)\}$ for all $t \in [0,T]$.*

It is essential to emphasize here that the learned operator $\mathcal{S}^*$ has to generate the *entire solution trajectory for* (1)*, given only the initial datum (and boundary conditions)*, as this is what the underlying solution operator $\mathcal{S}$ (and any numerical approximation to it) does.

Model Architecture. The backbone for the POSEIDON foundation model is provided by scOT or *scalable Operator Transformer*, see Figure 2 (a-c) for an illustrated summary. scOT is a *hierarchical multiscale vision transformer with lead-time conditioning* that processes lead time t and function space valued initial data input a to approximate the solution operator $\mathcal{S}(t,a)$ of the PDE (1).

For simplicity of exposition, we set $d = 2$ and $D = [0,1]^2$ as the underlying domain. As in a vision transformer [14], any underlying input is first *partitioned into patches and (linearly) embedded into a latent space*. At the level of function inputs $a \in C(D;\mathbb{R}^n)$, this amounts to the action of the *patch partitioning and embedding* operator $\mathbf{v} = \hat{\mathbf{E}}(a)$, with $\hat{\mathbf{E}}$ defined in **SM** (12). This operator transforms the input function into a piecewise constant function, which is constant within patches (subdivisions of the domain D), by taking weighted averages and then transforming these piecewise constant values into a C-dimensional latent space resulting in output $\mathbf{v} \in C(D;\mathbb{R}^C)$. In practice, a discrete version of this operator is used and is described in **SM** A.2.

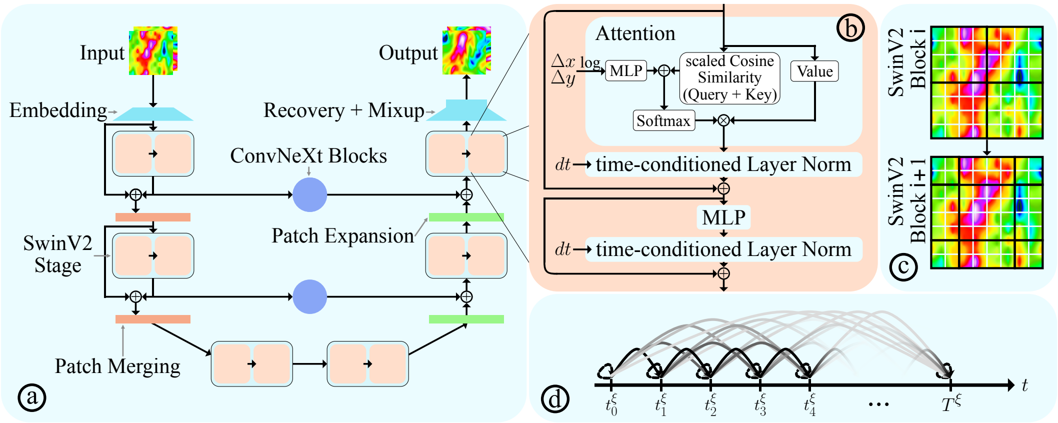


Figure 2: (a) scOT, the model underlying POSEIDON; (b) SwinV2 Transformer block; (c) Shifting Window over patch-based tokens with window (patch) boundaries with black (white); (d) all2all Training for time-dependent PDEs.

As shown in Figure 2 (a), this patch embedded output is then processed through a sequence of *SwinV2 transformer* blocks [38, 37], each of which has the structure of $SW_\ell : C(D;\mathbb{R}^C) \mapsto C(D;\mathbb{R}^C)$,

$$\begin{aligned} \mathbf{v}_\ell = SW_\ell(\mathbf{v}_{\ell-1}) &= \mathbf{v}'_\ell + LN_{\alpha_2^\ell,\beta_2^\ell}(MLP(\mathbf{v}'_\ell)), \\ \mathbf{v}'_\ell &= \mathbf{v}_{\ell-1} + LN_{\alpha_1^\ell,\beta_1^\ell}(W-MSA(\mathbf{v}_{\ell-1})). \end{aligned} \tag{3}$$

for layer index $\ell = 1, ..., L$. The main building block of a SwinV2 transformer block (3) (see Figure 2 (b)) is the *windowed multi-head self attention* operator defined in **SM** (14) (see **SM** A.2 for its discrete version). In particular, the attention operator acts only inside each window, which is defined by another (coarser) sub-division of D (see Figure 2 (c)), making it more computationally efficient than a standard vision transformer [14]. Moreover, the windows are shifted across layers, as depicted in Figure 2 (c), so that all the points in the domain can be attended to, by iteratively shifting windows across multiple layers, see **SM** A.2 for a detailed description of the SwinV2 block.

The MLP in (3) is defined by **SM** (15). We follow [55] to propose a *time-conditioning* strategy by introducing a *lead-time conditioned* layer norm in (3),

$$\begin{aligned} LN_{\alpha(t),\beta(t)}(\mathbf{v})(x) &= \alpha(t) \odot \frac{\mathbf{v}(x) - \mu_{\mathbf{v}}(x)}{\sigma_{\mathbf{v}}(x)} + \beta(t), \\ \mu_{\mathbf{v}}(x) &= \frac{1}{C}\sum_{j=1}^{C} \mathbf{v}_j(x),\ \sigma^2_{\mathbf{v}}(x) = \frac{1}{C}\sum_{j=1}^{C} (\mathbf{v}_j(x) - \mu_{\mathbf{v}}(x))^2, \end{aligned} \tag{4}$$

Here, $\alpha(t) = \alpha t + \overline{\alpha}$ and $\beta(t) = \beta t + \overline{\beta}$, with learnable $\alpha, \overline{\alpha}, \beta, \overline{\beta}$ although more general (small) MLPs can also be considered. This choice of time embedding enables *continuous-in-time evaluations*.

Finally, as depicted in Figure 2 (a), the SwinV2 transformer blocks (3) are arranged in a hierarchical, multiscale manner, within a U-Net style *encoder-decoder* architecture [11], by employing patch merging (downscaling) and patch expansion (upscaling) operations, (see **SM** A.2 for a detailed description). Moreover, layers at the same scale, but within the encoder and decoder stages of scOT, respectively, are connected through *ConvNeXt* convolutional layers [39], specified in **SM** A.2.

Training and Inference. We denote scOT by $\mathcal{S}^*_\theta : [0,T] \times \mathcal{X} \mapsto \mathcal{X}$, with trainable parameters $\theta \in \Theta \subset \mathbb{R}^p$. For scOT to approximate the solution operator $\mathcal{S}$ of (1), the parameters θ need to be determined by minimizing the mismatch between the predictions of scOT and ground truth training data, given in the form of trajectories $\{\mathcal{S}(t_k, a_i)\}$, for $0 \le k \le K$ and $1 \le i \le M$, with $a_i \sim \mu$ and $0 = t_0 < t_1 < \ldots t_k < \ldots < t_K = T$, being the time points at which the data is sampled. We assume that the data is sampled at the same timepoints for each sample a_i for simplicity. For training, it is natural to consider the loss function,

$$\overline{\mathcal{L}}(\theta) := \frac{1}{M(K+1)} \sum_{i=1}^{M}\sum_{k=0}^{K} \|\mathcal{S}^*_\theta(t_k, a_i) - \mathcal{S}(t_k, a_i)\|^p_{L^p(D)}, \tag{5}$$

with the (spatial) integral in (5) being replaced by a quadrature at some underlying sampling points and $p = 1$ in our paper. Thus, we use $K + 1$ samples per trajectory in order to train our model.

Given the fact that scaling up available training data is necessary for successful foundation models [23], we propose a *novel training strategy* that further leverages the *structure* of the time-dependent PDE (1) to increase the amount of training data. To this end, we consider the modified loss function,

$$\widehat{\mathcal{L}}(\theta) := \frac{1}{M\widehat{K}} \sum_{i=1}^{M} \sum_{k,\bar{k}=0,k\leq\bar{k}}^{K} \|\mathcal{S}(t_{\bar{k}} - t_k, u_i(t_k)) - \mathcal{S}^*_\theta(t_{\bar{k}} - t_k, u_i(t_k))\|^p_{L^p(D)}, \tag{6}$$

with $u_i(t_k) = \mathcal{S}(t_k, a_i)$ (approximately) solving (1) and $\widehat{K} = \frac{(K+1)(K+2)}{2}$. In other words, we leverage the fact that the solution operator of (1) possesses a *semi-group property* and one can realize,

$$u(t^*) = \mathcal{S}(t^*, a) = \mathcal{S}(t^* - t, u(t)) = \mathcal{S}(t^* - t, \mathcal{S}(t, a)), \ \forall\, 0 \leq t \leq t^* \leq T, \tag{7}$$

and any initial condition a. We term this use of all possible data pairs $(u(t_k), u(t_{\bar{k}}))$ with $k \leq \bar{k}$, see Figure 2 (d) for a visual representation, within a trajectory as *all2all training* and observe that it allows us to utilize *quadratic* $\mathcal{O}(K^2)$ samples per trajectory, when compared to the linear K samples used for training corresponding to the *vanilla* loss function (5). In practice, we consider a relative form of Equation 6 to balance out different scales of different operator outputs, see **SM** C for details.

Once scOT has been trained with (stochastic) gradient descent to find a (local) minimum θ^* of the all2all loss function (6), the trained model, denoted as $\mathcal{S}^*_{\theta^*}$ can be deployed for inference for any initial condition $a \in \mathcal{X}$ and for any $t \in \mathbb{R}_+$ by directly applying $\mathcal{S}^*_{\theta^*}(t, a)$ to provide continuous-in-time evaluation of the entire trajectory. However, it might be advantageous to infer using *autoregressive rollouts* [36]. To this end, we consider a sequence $0 = t^*_0 < t^*_1 < \ldots < t^*_\kappa = t$. Then, the rollout,

$$\mathcal{S}(t, a) \approx \mathcal{S}^*_{\theta^*}\left(t^*_\kappa - t^*_{\kappa-1}, \mathcal{S}^*_{\theta^*}(\ldots\ldots\mathcal{S}^*_{\theta^*}\left(t^*_2 - t^*_1, \mathcal{S}^*_{\theta^*}(t^*_1, a)\right))\right), \tag{8}$$

of κ successive applications of the trained scOT approximates the solution operator at any time t.

Pretraining. The key point in the development of any foundation model is the *pretraining* step, in which the model is trained on a diverse set of data distributions, rather than just on data drawn from one specific operator. To formulate pretraining and subsequent steps precisely, we introduce index sets Λ, Ξ and let $\lambda \in \Lambda$ and $\xi \in \Xi$ correspond to indexing the PDE type and the data-distribution, respectively. To see this, we fix any $\lambda \in \Lambda, \xi \in \Xi$ and tag the differential and boundary operators $\mathcal{L}, \mathcal{B}$ in the PDE (1) by $\mathcal{L}^\lambda$ and $\mathcal{B}^\lambda$. Similarly the initial distribution μ in (1) is tagged by μ^ξ and the resulting solution operator for PDE (1) with $\mathcal{L}^\lambda, \mathcal{B}^\lambda$ and initial datum $a \sim \mu^\xi$ is denoted by $\mathcal{S}^{\lambda,\xi}$. In other words, Λ, Ξ indexes the entire set of PDEs and data distributions that we consider.

Next, we fix index sets, $\widehat{\Lambda} \subset \Lambda$ and $\widehat{\Xi} \subset \Xi$ and consider a set of PDEs (1), indexed by $\lambda \in \widehat{\Lambda}$ and with data distributions μ^ξ, indexed by $\xi \in \widehat{\Xi}$ as the *pretraining dataset*, which consists of the corresponding trajectories, $\{\mathcal{S}^{\lambda,\xi}(t, \cdot)\}$, for all t and all $(\lambda, \xi) \in (\widehat{\Lambda}, \widehat{\Xi})$.

Let $n^{\widehat{\Xi}}$ be the maximum number of components of the solution vectors for all the operators in the pretraining dataset. By including additional (constant 0 over space and time) components, we augment the relevant solution operators (for which the number of components is below $n^{\widehat{\Xi}}$) such that for each $\lambda \in \widehat{\Lambda}, \xi \in \widehat{\Xi}$, all the input functions have the same number of $n^{\widehat{\Xi}}$ components (channels). These inputs are fed into a scOT model $\Pi^{\widehat{\Lambda},\widehat{\Xi}}_\theta : [0, \widehat{T}] \times L^p(D; \mathbb{R}^{n^{\widehat{\Xi}}}) \mapsto C([0, \widehat{T}]; L^p(D; \mathbb{R}^{n^{\widehat{\Xi}}}))$, with $\widehat{T}$ being the supremum over all the final times in the pretraining dataset. The trainable parameters θ of this pretrained model are then determined by *minimizing the mismatch between model predictions and ground truth over all PDEs and data distributions in the pretraining dataset* resulting in,

$$\Pi^{\widehat{\Lambda},\widehat{\Xi}}_* = \Pi^{\widehat{\Lambda},\widehat{\Xi}}_{\theta^*}, \text{ with } \quad \theta_* = \text{argmin}_{\theta\in\Theta} \frac{1}{|\widehat{\Lambda}||\widehat{\Xi}|} \sum_{\lambda\in\widehat{\Lambda}} \sum_{\xi\in\widehat{\Xi}} \widehat{\mathcal{L}}^{\lambda,\xi}(\theta) \,, \tag{9}$$

with $\widehat{\mathcal{L}}^{\lambda,\xi}$ obtained by replacing $\mathcal{S}$ and $\mathcal{S}^*_\theta$ in (6) with $\mathcal{S}^{\lambda,\xi}$ and $\Pi^{\widehat{\Lambda},\widehat{\Xi}}_\theta$, respectively.

Finetuning. To *finetune* the pretrained foundation model $\Pi^{\widehat{\Lambda},\widehat{\Xi}}_*$ for any downstream task, corresponding any specific solution operator $\mathcal{S}^{\lambda,\xi}$ for any $\lambda \in \Lambda, \xi \in \Xi$, we decompose the vector of learnable parameters $\theta \in \Theta \subset \mathbb{R}^p$ as $\theta = [\widehat{\theta}, \widetilde{\theta}, \widetilde{\theta}^{\mathcal{N}}]$, with $\widehat{\theta} \in \mathbb{R}^{\hat{p}}$, $\widetilde{\theta} \in \mathbb{R}^{\tilde{p}}$, and $\widetilde{\theta}^{\mathcal{N}} \in \mathbb{R}^{\tilde{p}_{\mathcal{N}}}$ and $\hat{p}+\tilde{p}+\tilde{p}_{\mathcal{N}} = p$, with $\tilde{p}, \tilde{p}_{\mathcal{N}} \ll \hat{p}$. A gradient descent step for finetuning is then written as,

$$\begin{aligned} \forall r \geq 1, \quad & [\widehat{\theta}_{r+1}, \widetilde{\theta}_{r+1}, \widetilde{\theta}^{\mathcal{N}}_{r+1}] = [\widehat{\theta}_r, \widetilde{\theta}_r, \widetilde{\theta}^{\mathcal{N}}_r] - [\widehat{\eta}_r, \widetilde{\eta}_r, \widetilde{\eta}^{\mathcal{N}}_r] \nabla_\theta \widehat{\mathcal{L}}^{\lambda,\xi}(\theta_r), \\ & \widehat{\theta}_0 = \widehat{\theta}_*, \quad \widetilde{\theta}^{\mathcal{N}}_0 = \widetilde{\theta}^{\mathcal{N}}_*, \quad \widetilde{\theta}_0 \sim \widetilde{P}, \quad \widetilde{P} \in \mathrm{Prob}(\mathbb{R}^{\tilde{p}}). \end{aligned} \tag{10}$$

Hence, during finetuning, a subset of parameters $\widetilde{\theta}$ of the foundation model are trained from scratch with random initializations, whereas the complementary, much larger subset of $\widehat{\theta}$ and $\widetilde{\theta}_{\mathcal{N}}$ is initialized by *transferring* the corresponding parameters from the pretrained model. When $\lambda \notin \widehat{\Lambda}$, $\widetilde{\theta}$ consists of the *embedding/recovery* parameters. On the other hand, if $\lambda \in \widehat{\Lambda}$, then all trainable parameters, including the patch embeddings/recovery, are initialized with the corresponding parameters of the pretrained model. However, the corresponding learning rate $\widetilde{\eta}_r \gg \widehat{\eta}_r$ in (10) is much higher. Similarly, the time embeddings $\widetilde{\theta}^{\mathcal{N}}$, i.e., the trainable parameters in the layer-norm operators (4) are always initialized from the corresponding time embeddings in the pretrained model but finetuned with a higher learning rate $\widetilde{\eta}^{\mathcal{N}}$.

3 Experiments

Pretraining Dataset. We pretrain POSEIDON on a dataset containing 6 operators, defined on the space-time domain $[0,1]^2 \times [0,1]$. 4 of these operators (CE-RP, CE-KH, CE-CRP, CE-Gauss) pertain to the compressible Euler equations (**SM** (37)) of gas dynamics and 2 (NS-Sines, NS-Gauss) to the incompressible Navier-Stokes equations (**SM** (31)) of fluid dynamics, see **SM** Table 3 for abbreviations and **SM** B.1 for a detailed description of these datasets. These datasets have been selected to highlight different aspects of the PDEs governing fluid flows (shocks and shear layers, global and local turbulent features, and mixing layers etc.). The pretraining dataset contains 9640 and 19640 trajectories for the Euler and Navier-Stokes operators, respectively, leading to a total of 77840 trajectories. Each trajectory is uniformly sampled at 11 time snapshots. Within the all2all training procedure (Section 2), this implies a total of 66 input-output pairs per trajectory, leading to approx 5.11M training examples in the pretraining dataset.

Downstream Tasks. To evaluate POSEIDON (and the baselines), we select a suite of 15 challenging downstream tasks, see **SM** Table 4 for abbreviations and **SM** B.2 for detailed description. Each of these tasks is a (variant of) well-known benchmarks for PDEs in the numerical analysis and computational physics literature and corresponds to a distinct PDE solution operator. They have also been selected for their diversity in terms of the PDE types as they contain linear (4) and nonlinear (11), time-dependent (12) and time-independent (3), elliptic (2), parabolic (1), hyperbolic (4) and mixed-type (8). The tasks also cover a wide gamut of physical processes across a range of spatio-temporal scales. Moreover, we emphasize that each of the downstream tasks is *out-of-distribution* with respect to the pretraining data. While 6 of them do pertain to the Euler and Navier-Stokes equations seen in the pretraining dataset but with very different data distributions, the remaining 9 involve PDEs not seen during pretraining. These include 3 (NS-Tracer-PwC, FNS-KF, GCE-RT) which add new physical processes (tracer transport, forcing, gravity) to the Navier-Stokes and Euler equations. 3 more (Wave-Gauss, Wave-Layer, ACE) involve completely new time-dependent PDEs (Wave Eqn., Allen-Cahn Eqn.) and the final 3 (SE-AF, Poisson-Gauss, Helmholtz) *even consider time-independent PDEs*, which is in stark contrast to the pretraining dataset where only 2 time-dependent PDEs are covered. For these steady state PDEs, we finetune them by using the interpretation of the PDE (2) as a *long-time limit* of the time-dependent PDE (1) with a normalized lead time of 1. Finally, the tasks have also been selected to probe the ability of the foundation model to handle different *task or operator types.* To this end, we point out that all the operators in the pretraining dataset simply map the initial conditions to the solution at later times in time-dependent fluid flows on the two-dimensional unit square with *periodic boundary conditions.* While some of the downstream tasks (8 out of 15) do pertain to this type of operators, the remaining (7 out of 15) tasks involve different types of operators which include operators mapping the coefficients or PDE parameters to the PDE solution (5 out of 15), forcing term to the PDE solution (2) and domain shape to the PDE solution. Moreover, many of the downstream tasks are with non-periodic boundary conditions while one of them is even on a non-Cartesian domain. Thus, these downstream tasks deviate from the setup of the pretraining operators and provide a hierarchy of challenges for any foundation model.

Models and Baselines. We consider three different POSEIDON models: i) POSEIDON-T with ≈ 21M parameters, ii) POSEIDON-B with ≈ 158M parameters, and iii) POSEIDON-L with ≈ 629M parameters. The detailed specifications of each of these models is provided in **SM** C.1. As baselines, in addition to the standalone scOT, we use trained from scratch neural operators in the form of the widely used FNO [33] and recently proposed CNO [60], each augmented with time-conditioned instance normalizations. Foundation model baselines are provided by MPP-aVIT (MPP) [49] and we also pretrain a CNO [60] model (see details in **SM** C.5) on our pretraining dataset, resulting in an additional foundation model baseline termed CNO-FM, see **SM** C for details on baselines.

Evaluation Metrics. All the models and baselines are evaluated on each task in terms of the relative L^1 error at the underlying final time. This choice is motivated by the fact that errors tend to grow over time, making final time prediction harder than any time-averaged quantities, see **SM** D.6.3. This also corresponds well to the interpretation of time-independent PDEs as long-time limits of (1). Following [23] that advocates this approach for LLMs, we evaluate all models in terms of *scaling* curves which plot the test error for each task vs. the number of task-specific training examples, see **SM** D.1. To extract further information from scaling plots, we introduce two evaluation metrics,

$$\mathbf{AG}_{\mathrm{S}}(\mathrm{model}) := \frac{\mathcal{E}_S(\mathrm{FNO})}{\mathcal{E}_S(\mathrm{model})}, \quad \mathbf{EG}_{\mathrm{S}}(\mathrm{model}) := \frac{S}{s}, \text{ where } \mathcal{E}_s(\mathrm{model}) = \mathcal{E}_S(\mathrm{FNO}), \qquad (11)$$

with $\mathcal{E}_S(\mathrm{model})$ being the relative error (at final time) for the model with S trajectories. Thus, *Accuracy Gain* $\mathbf{AG}_S$ measures how accurate the model is w.r.t. FNO for a given number (S) of samples while *Efficiency Gain* $\mathbf{EG}_S$ measures how much fewer (greater) number of samples the model needs to attain the same error level as FNO trained on S samples. **AG** is the relevant metric for the *limited compute* regime whereas **EG** is relevant for the *limited data* regime.

POSEIDON performs very well on all downstream tasks. From the scaling plots **SM** Figures 7 to 21, we observe that POSEIDON readily outperforms FNO on *all the 15 downstream tasks*. This point is further buttressed by Table 1, where the **EG** and **AG** (11) metrics are presented (see also **SM** Table 8 for these metrics for the POSEIDON-B and -T models). We observe from this table that POSEIDON requires *far fewer* task specific samples to attain the same error level as FNO does with $S = 1024$ samples for time-dependent PDEs ($S = 4096$ for time-independent PDEs). In fact, there are 4 tasks for which a mere 3 task-specific samples suffice for POSEIDON to attain the same error as FNO with 1024 samples. From **SM** Table 9, we observe that, on an average (median), only 20 samples are needed for POSEIDON-L to reach the errors of FNO with 1024 samples and in 13 (of the 15) tasks, POSEIDON-L needs an order of magnitude fewer samples than FNO. Similarly from Table 1 and **SM** Table 9, we see that for the same number ($S = 128$ for time-dependent, and $S = 512$ for time-independent PDEs) of samples, POSEIDON-L has significantly lower error than FNO, with gains ranging from anywhere between 10% to a factor of 25, with the mean gain of accuracy being an *entire order of magnitude*.

Among the trained-from-scratch neural operator baselines, CNO and scOT are comparable in performance to each other, while both outperform FNO significantly on almost all tasks (see Table 1 and **SM** Table 9). However, POSEIDON is much superior to both of them, in terms of gains in sample efficiency (median gain of an order of magnitude) as well as accuracy (average gain of a factor of 4).

POSEIDON generalizes well to unseen physics. This impressive performance of POSEIDON is particularly noteworthy as all the downstream tasks are *out-of-distribution* with respect to the pretraining dataset. This performance is also consistent across the 9 tasks which involve PDEs not seen during pretraining. POSEIDON is the best performing model on 8 of these tasks, including all the time-dependent PDEs. It is only for 1 of the time-indepedent PDEs, which constitute the hardest generalization challenge, that POSEIDON is outperformed by CNO, but only marginally. These results underscore the ability of POSEIDON to learn completely new physical processes and contexts from a few downstream task-specific samples.

Architecture of the foundation model matters. We observe from **SM** D.1 and Table 1 (see also **SM** Table 9) that POSEIDON outperforms CNO-FM clearly on 14 out of 15 downstream tasks. On average (median over all tasks), CNO-FM requires approximately 100 task-specific examples to attain the error levels of FNO with 1024 samples, whereas POSEIDON only requires approximately 20. As CNO-FM and POSEIDON have been pretrained on exactly the same dataset, this difference in performance can be largely attributed to architectural differences as CNO-FM is based on multiscale CNNs, in contrast to the multiscale vision transformer which is the backbone of POSEIDON.

Table 1: Efficiency gain EG ((11) with $S = 1024$ for time-dependent and $S = 4096$ for time-independent PDEs) and Accuracy Gain (*AG*) ((11) with $S = 128$ for time-dependent and $S = 512$ for time-independent PDEs) for all models and downstream tasks.

	Pretrained Models						Models trained from Scratch					
	Poseidon-L		CNO-FM		MPP-B		CNO		scOT		FNO	
	EG	*AG*	EG	*AG*	EG	*AG*	EG	*AG*	EG	*AG*	EG	*AG*
NS-PwC	**890.6**	***24.7***	16.6	*3.3*	7.4	*2.3*	3.7	*1.5*	5.4	*2.0*	1	*1*
NS-SVS	**502.9**	***7.3***	59.6	*3.1*	34.8	*2.2*	73.2	*3.4*	10.2	*1.2*	1	*1*
NS-BB	**552.5**	***29.3***	10.6	*3.9*	4.6	*2.6*	2.7	*1.7*	3.4	*2.1*	1	*1*
NS-SL	**21.9**	***5.5***	0.4	*0.8*	0.3	*0.8*	0.8	*1.2*	0.3	*0.8*	1	*1*
NS-Tracer-PwC	**49.8**	***8.7***	17.8	*3.6*	8.5	*2.7*	4.6	*1.9*	4.6	*1.9*	1	*1*
FNS-KF	**62.5**	***7.4***	13.2	*2.7*	2.0	*1.6*	3.1	*1.5*	3.3	*0.9*	1	*1*
CE-RPUI	**352.2**	***6.5***	33.2	*2.3*	0.0	*1.2*	12.5	*1.8*	15.6	*2.1*	1	*1*
CE-RM	**4.6**	***1.2***	0.6	*1.0*	0.0	*0.2*	1.7	*1.1*	0.4	*1.0*	1	*1*
SE-AF	3.4	*1.2*	4.8	*1.3*	2.2	*1.1*	**5.5**	***1.5***	1.2	*1.0*	1	*1*
GCE-RT	**5.3**	***2.0***	1.2	*1.0*	0.0	*0.3*	1.2	*1.4*	1.1	*1.1*	1	*1*
Wave-Layer	**46.5**	***6.1***	5.6	*2.2*	0.0	*0.9*	11.4	*3.0*	13.0	*2.9*	1	*1*
Wave-Gauss	**62.1**	***5.6***	6.0	*1.8*	0.0	*0.8*	14.0	*2.6*	9.2	*2.1*	1	*1*
ACE	**17.0**	***11.6***	1.7	*2.0*	0.0	*0.3*	4.5	*4.6*	6.5	*5.2*	1	*1*
Poisson-Gauss	**42.5**	***20.5***	25.0	*9.2*	17.0	*7.3*	21.1	*7.0*	9.8	*5.3*	1	*1*
Helmholtz	**78.3**	*6.1*	54.0	*5.1*	22.4	*3.0*	68.9	*7.3*	60.4	***9.0***	1	*1*

The second baseline foundation model, MPP-B of [49], is based on a transformer with axial attention and is pretrained on the PDEBench dataset [71]. However, it has been trained to predict the next time step, given a context window of τ previous time steps, with $\tau = 16$ as the default. We emphasize that this next step prediction, given a context window, *does not solve the underlying operator learning task* **OLT** directly as **OLT** requires that the entire trajectory needs to be generated, given the initial data. Hence, we had to finetune the pretrained MPP model with varying context windows (starting with window size of 1), see **SM** C.6 for details. We see from Table 1 and **SM** Table 9 that the finetuned MPP modestly outperformed FNO on some (8 out of 15) of the downstream tasks but it failed on the rest of them, where MPP simply could not attain the error levels of FNO, as it did not converge or even blew up with increasing number of downstream samples (see scaling plots in **SM** D.1).

In this context, it can be argued that the Poseidon-L model is larger in size than both CNO-FM and MPP-B and perhaps, it is this size difference which explains the differential in performance. However, this is far from the case. As shown in all the scaling plots of **SM** D.1 and **SM** Tables 8 and 9, both CNO-FM and MPP-B are significantly inferior to the Poseidon-B model, which is comparable in size. In fact, we can see from these tables that even the Poseidon-T model, which is an order of magnitude smaller in size, outperforms CNO-FM and MPP-B handily. It also readily outperforms all the trained-from-scratch neural operators (CNO, FNO and scOT) which are of comparable size to it, leading us to conclude that it is the combination of the pretraining dataset as well as the underlying architecture, rather than just model size, that underpins the superior performance of Poseidon.

Poseidon scales with model size. Nevertheless, the model size of Poseidon does matter. As seen from **SM** Figure 22, both the training as well as evaluation (validation) errors on the pretraining dataset clearly decrease with increasing model size of Poseidon. However, does this scaling with model size lead to any impact on the performance of these models, when finetuned on downstream tasks? We see from the scaling plots in **SM** D.1 that Poseidon-L consistently outperforms the smaller Poseidon-B on most downstream tasks. This trend is reinforced by **SM** Tables 8 and 9, where we find that, on an average, increasing model size correlates with a consistent decrease in test error as well as an increase in sample efficiency of the pretrained model on downstream tasks.

POSEIDON scales with dataset size. In **SM** Figure 23, we show how by increasing the size of the pretraining dataset, in terms of the number of trajectories, the training and validation losses for the pretrained POSEIDON-B model decrease. Moreover, from **SM** Figures 24 to 38, where we plot the test error versus number of downstream task-specific samples for 2 different models, POSEIDON-B trained on the full pretraining dataset and on one-eighth of the pretraining dataset, we find that for most (9 of the 15) of the downstream tasks, increasing the number of samples in the pretraining dataset, by an order of magnitude, *does lead to significantly greater accuracy* even at the downstream task level. For the remaining tasks, the models trained with less data are either on par or marginally inferior to the model trained with the full dataset.

The quality/diversity of the pretraining dataset matters. To demonstrate this point, we consider two different datasets: one in which half the trajectories of the pretraining dataset for POSEIDON-B are randomly dropped (from every operator), and the other where less diversity of the pretraining dataset is imposed by dropping all the trajectories corresponding to 3 out of 6 operators, namely CE-CRP, CE-Gauss and NS-Sines. Thus, the total size of both datasets is the same but one is clearly less diverse than the other. The respective POSEIDON-B models are then evaluated on all the downstream tasks. As shown **SM** Figures 24 to 38, the model trained on less diverse data performs worse than its counterpart on 10 out of the 15 tasks and is on par on 4 of them. Thus, we demonstrate that in a large majority of downstream tasks, the quality/diversity of the pretraining dataset matters.

How does POSEIDON generalize to unseen physics? In order to understand the *surprising* ability of POSEIDON to generalize so well to unseen and *a priori* unrelated PDEs and physical processes downstream, we present three case studies in **SM** D.4 to uncover some of the inner workings of this foundation model. In particular, we first consider the CE-RPUI downstream task. This task pertains to the compressible Euler equations, which are included in the pretraining dataset. However, the underlying initial data distribution is not seen during pretraining, making the task *out-of-distribution*. We show in **SM** D.4.1, how POSEIDON leverages different features of different operators from the pretraining dataset to learn this task accurately with very few samples (see **SM** Figure 39). In particular, the diversity of the pretraining dataset is more instrumental in ensuring better generalization to this unseen initial condition than the size of the dataset.

In **SM** D.4.3, we study the Poisson-Gauss task to understand arguably the most surprising finding about the POSEIDON foundation models, i.e., their ability to generalize well to PDEs that are completely unrelated to the Euler and Navier-Stokes equations of fluid dynamics. This task pertains to the Poisson equation (68) with a forcing term, which is a superposition of Gaussians. The task is very different from those seen during pretraining in multiple ways, namely the underlying PDE is not only time-independent (in contrast to the time-dependent PDEs of pretraining) but also elliptic (whereas the PDEs during pretraining are either hyperbolic or convection-dominated) and the boundary conditions are Dirichlet (instead of Periodic) leading to very different physics, that of diffusion and smoothing, being manifested for this task, when contrasted with the physics seen during pretraining which is dominated by transport, shock wave propagation and fluid mixing. Given this context, one would not expect POSEIDON to perform well on this task. Yet, from **SM** Figures 20 and 74, we know that POSEIDON performs exceptionally well, learning the solution operator accurately with a few samples. As we elaborate in **SM** D.4.3, POSEIDON does not use the first few training examples to *forget* the physics that it has learned during pretraining and learn the new physics for this task after that. Rather surprisingly, as illustrated in **SM** Figure 43, already with *one* task specific training example, POSEIDON outputs an (approximation of the) input, rather than the expected dynamic evolution of fluids with Gaussian inputs (see **SM** Figures 56 and 60) seen during pretraining. Then, with very few (16) examples, it is able to learn the rudiments of diffusion and smoothing of features (**SM** Figure 43), which are characteristics of elliptic equations. To further test how the foundation model leverages physics representations learned during pretraining, we *froze* the latent space by only finetuning the embeddings and freezing the latent space parameters by setting $\widehat{\theta}_r = \widehat{\theta}_*$ for all r, in (10) for finetuning. As shown in (**SM** Figure 44), even this *frozen latent* version of POSEIDON is very effective at learning the underlying solution operator, demonstrating that very rich physical representations were learned during pretraining.

Further results on the robustness of POSEIDON for different factors and ablations as well as comparisons with other foundation models is provided in **SM** D and details of computational resources are described in **SM** E.

4 Discussion

Summary. In this paper, we have presented POSEIDON, a family of foundation models for learning PDEs. The backbone of POSEIDON is scOT, a multiscale vision transformer with shifted-windowed (SwinV2) attention that maps input functions (initial data, coefficients, sources) etc. to the solution (trajectory) of a PDE. Lead-time conditioning through a time-modulated layer norm allows for continuous-in-time evaluation and a novel all2all training strategy enables the scaling up of training data by leveraging the semi-group structure of solutions of time-dependent PDEs. POSEIDON is pretrained on a diverse large-scale dataset of operators for the compressible Euler and incompressible Navier-Stokes PDEs. Its performance is evaluated on a challenging suite of 15 *out-of-distribution* downstream tasks covering a wide variety of PDEs and data distributions. POSEIDON displays excellent downstream performance and is the best performing model on 14 of the 15 tasks. In particular, it requires orders of magnitude (median of 50) fewer task-specific samples to attain the same error as the widely used FNO. This large gain in sample efficiency as well as order of magnitude gains in accuracy also holds for PDEs that are not seen during pretraining, making us conclude that POSEIDON generalizes well to *new physics*. POSEIDON also scales with model and dataset size, with respect to pretraining and even downstream task performance. To the best of our knowledge, this is the first time that it has been clearly demonstrated that by pretraining on a very small set of PDEs, a foundation model can generalize to a wider variety of unseen and unrelated PDEs and data distributions downstream. Thus, we provide an affirmative answer to the very fundamental question of whether foundation models for PDEs are even feasible. Moreover, we investigate possible mechanisms via which POSEIDON can effectively leverage representations, learnt during pretraining, to accurately learn downstream tasks by finetuning on a few task-specific examples. Our case studies suggest hitherto undiscovered relationships between different PDEs that enable this transfer to materialize. Finally, all the models are made publicly available, as well as the pretraining and downstream datasets are open sourced in the PDEGYM collection.

Related Work. Foundation models for PDEs are of very recent vintage. The foundation model of [67] is limited to very specific elliptic Poisson and Helmholtz PDEs with a FNO backbone whereas ICON [74] considers a very small 1-D dataset. Neither of these models are comparable in scope to POSEIDON. Universal physics transformers [1] employs transformers but its focus is on incompressible fluid flows and the ability to generalize across Eulerian and Lagrangian data. Thus, a direct comparison with POSEIDON is not possible. On the other hand, MPP [49] and DPOT [19] are designed to be general purpose foundation models for PDEs that can be compared to POSEIDON. We have already extensively compared MPP with POSEIDON in Section 3 to demonstrate the very large superiority of POSEIDON across various metrics. Although DPOT has a different architecture (Adaptive FNO) and was trained on more datasets than MPP, it follows a similar training and evaluation strategy of next time-step prediction, given a context window of previous time-steps. As argued before, this does not solve the operator learning task of generating the entire trajectory, given initial data. At the time of writing this paper, DPOT was not publicly available but it was released by the time this paper has been revised, enabling us to modify the fine-tuning procedure of DPOT and to perform comparisons between it and POSEIDON. While directing the interested reader to **SM** D.5 for details, we summarize our findings by observing that POSEIDON models are significantly better performing than DPOT foundation models, both in terms of accuracy and sample efficiency.

Limitations. The range of PDEs and underlying data distributions is huge and POSEIDON was only trained and evaluated on a few of them. Although the results here clearly demonstrate its ability to learn unseen physics from a few task-specific training examples, we anticipate that given that it is scaling with respect to both data quantity and quality, POSEIDON's performance as a general purpose PDE foundation model will significantly improve when it is pretrained with even more diverse PDE datasets in the future. In particular, pretraining with time-independent PDEs (particularly elliptic PDEs) as well as a larger range of time-scales in time-dependent PDEs will greatly enhance POSEIDON. The focus here was on Cartesian geometries although POSEIDON displayed the ability to generalize to non-Cartesian geometries, via masking, on the SE-AF task. We plan to add several non-Cartesian examples in the pretraining dataset to augment POSEIDON's performance on general geometries/boundary conditions. Moreover, given the fact that POSEIDON serves as a fast and accurate neural PDE surrogate, its extension to qualitatively different downstream tasks such as uncertainty quantification [45], inverse problems [53] and PDE-constrained optimization [46] is fairly straightforward and will be considered in future work.

Acknowledgments and Disclosure of Funding

This work was supported by a grant from the Swiss National Supercomputing Centre (CSCS) under project ID 1217.

References

[1] B. Alkin, A. Fürst, S. Schmid, L. Gruber, M. Holzleitner, and J. Brandstetter. Universal physics transformers: A framework for efficiently scaling neural operators, 2024.

[2] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization, 2016.

[3] F. Bartolucci, E. de Bézenac, B. Raonic, R. Molinaro, S. Mishra, and R. Alaifari. Representation equivalent neural operators: a framework for alias-free operator learning. *arXiv:2305.19913*, 2023.

[4] I. Batatia, P. Benner, Y. Chiang, A. M. Elena, D. P. Kovács, J. Riebesell, X. R. Advincula, M. Asta, M. Avaylon, W. J. Baldwin, F. Berger, N. Bernstein, A. Bhowmik, S. M. Blau, V. Cărare, J. P. Darby, S. De, F. D. Pia, V. L. Deringer, R. Elijošius, Z. El-Machachi, F. Falcioni, E. Fako, A. C. Ferrari, A. Genreith-Schriever, J. George, R. E. A. Goodall, C. P. Grey, P. Grigorev, S. Han, W. Handley, H. H. Heenen, K. Hermansson, C. Holm, J. Jaafar, S. Hofmann, K. S. Jakob, H. Jung, V. Kapil, A. D. Kaplan, N. Karimitari, J. R. Kermode, N. Kroupa, J. Kullgren, M. C. Kuner, D. Kuryla, G. Liepuoniute, J. T. Margraf, I.-B. Magdău, A. Michaelides, J. H. Moore, A. A. Naik, S. P. Niblett, S. W. Norwood, N. O'Neill, C. Ortner, K. A. Persson, K. Reuter, A. S. Rosen, L. L. Schaaf, C. Schran, B. X. Shi, E. Sivonxay, T. K. Stenczel, V. Svahn, C. Sutton, T. D. Swinburne, J. Tilly, C. van der Oord, E. Varga-Umbrich, T. Vegge, M. Vondrák, Y. Wang, W. C. Witt, F. Zills, and G. Csányi. A foundation model for atomistic materials chemistry, 2024.

[5] J. Bell, P. Collela, and H. M. Glaz. A second-order projection method for the incompressible Navier-Stokes equations. *J. Comput. Phys.*, 85:257–283, 1989.

[6] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, E. Brynjolfsson, S. Buch, D. Card, R. Castellon, N. Chatterji, A. Chen, K. Creel, J. Q. Davis, D. Demszky, C. Donahue, M. Doumbouya, E. Durmus, S. Ermon, J. Etchemendy, K. Ethayarajh, L. Fei-Fei, C. Finn, T. Gale, L. Gillespie, K. Goel, N. Goodman, S. Grossman, N. Guha, T. Hashimoto, P. Henderson, J. Hewitt, D. E. Ho, J. Hong, K. Hsu, J. Huang, T. Icard, S. Jain, D. Jurafsky, P. Kalluri, S. Karamcheti, G. Keeling, F. Khani, O. Khattab, P. W. Koh, M. Krass, R. Krishna, R. Kuditipudi, A. Kumar, F. Ladhak, M. Lee, T. Lee, J. Leskovec, I. Levent, X. L. Li, X. Li, T. Ma, A. Malik, C. D. Manning, S. Mirchandani, E. Mitchell, Z. Munyikwa, S. Nair, A. Narayan, D. Narayanan, B. Newman, A. Nie, J. C. Niebles, H. Nilforoshan, J. Nyarko, G. Ogut, L. Orr, I. Papadimitriou, J. S. Park, C. Piech, E. Portelance, C. Potts, A. Raghunathan, R. Reich, H. Ren, F. Rong, Y. Roohani, C. Ruiz, J. Ryan, C. Ré, D. Sadigh, S. Sagawa, K. Santhanam, A. Shih, K. Srinivasan, A. Tamkin, R. Taori, A. W. Thomas, F. Tramèr, R. E. Wang, W. Wang, B. Wu, J. Wu, Y. Wu, S. M. Xie, M. Yasunaga, J. You, M. Zaharia, M. Zhang, T. Zhang, X. Zhang, Y. Zhang, L. Zheng, K. Zhou, and P. Liang. On the opportunities and risks of foundation models, 2022.

[7] B. Bonev, T. Kurth, C. Hundt, J. Pathak, M. Baust, K. Kashinath, and A. Anandkumar. Spherical Fourier Neural Operators: Learning Stable Dynamics on the Sphere, June 2023. arXiv:2306.03838 [physics].

[8] J. Brandstetter, D. E. Worrall, and M. Welling. Message passing neural PDE solvers. In *International Conference on Learning Representations*, 2022.

[9] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, X. Chen, K. Choromanski, T. Ding, D. Driess, A. Dubey, C. Finn, P. Florence, C. Fu, M. G. Arenas, K. Gopalakrishnan, K. Han, K. Hausman, A. Herzog, J. Hsu, B. Ichter, A. Irpan, N. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, I. Leal, L. Lee, T.-W. E. Lee, S. Levine, Y. Lu, H. Michalewski, I. Mordatch, K. Pertsch, K. Rao, K. Reymann, M. Ryoo, G. Salazar, P. Sanketi, P. Sermanet, J. Singh, A. Singh, R. Soricut, H. Tran, V. Vanhoucke, Q. Vuong, A. Wahid, S. Welker, P. Wohlhart, J. Wu, F. Xia, T. Xiao, P. Xu, S. Xu, T. Yu, and B. Zitkovich. Rt-2: Vision-language-action models transfer web knowledge to robotic control, 2023.

[10] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners, 2020.

[11] H. Cao, Y. Wang, J. Chen, D. Jiang, X. Zhang, Q. Tian, and M. Wang. Swin-Unet: Unet-Like Pure Transformer for Medical Image Segmentation. In L. Karlinsky, T. Michaeli, and K. Nishino, editors, *Computer Vision – ECCV 2022 Workshops*, Lecture Notes in Computer Science, pages 205–218, Cham, 2023. Springer Nature Switzerland.

[12] S. Cao. Choose a transformer: Fourier or galerkin. In *35th conference on neural information processing systems*, 2021.

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

[14] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, June 2021. arXiv:2010.11929 [cs].

[15] L. C. Evans. *Partial differential equations*, volume 19. American Mathematical Soc., 2010.

[16] U. S. Fjordholm, R. Käppeli, S. Mishra, and E. Tadmor. Construction of approximate entropy measure valued solutions for hyperbolic systems of conservation laws. *Found. Comput. Math.*, 17(3):763–827, 2017.

[17] . Google Gemini Team. Gemini: A family of highly capable multimodal models, 2024.

[18] J. K. Gupta and J. Brandstetter. Towards multi-spatiotemporal-scale generalized pde modeling, 2022.

[19] Z. Hao, C. Su, S. Liu, J. Berner, C. Ying, H. Su, A. Anandkumar, J. Song, and J. Zhu. DPOT: Auto-Regressive Denoising Operator Transformer for Large-Scale PDE Pre-Training, Mar. 2024. arXiv:2403.03542 [cs, math].

[20] Z. Hao, Z. Wang, H. Su, C. Ying, Y. Dong, S. Liu, Z. Cheng, J. Song, and J. Zhu. Gnot: A general neural operator transformer for operator learning, 2023.

[21] D. Hendrycks and K. Gimpel. Gaussian error linear units (gelus), 2023.

[22] J. S. Hesthaven. *Numerical methods for conservation laws: From analysis to algorithms.* SIAM, 2018.

[23] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling Laws for Neural Language Models, Jan. 2020. arXiv:2001.08361 [cs, stat].

[24] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang. Physics informed machine learning. *Nature Reviews Physics*, pages 1–19, may 2021.

[25] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[26] G. Kissas, J. H. Seidman, L. F. Guilhoto, V. M. Preciado, G. J. Pappas, and P. Perdikaris. Learning operators with coupled attention. *Journal of Machine Learning Research*, 23(215):1–63, 2022.

[27] N. Kovachki, Z. Li, B. Liu, K. Azizzadensheli, K. Bhattacharya, A. Stuart, and A. Anandkumar. Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481v3*, 2021.

[28] R. Käppeli and S. Mishra. Well-balanced schemes for the euler equations with gravitation. *Journal of Computational Physics*, 259:199–219, 2014.

[29] L. D. Landau and E. M. Lipschitz. *Fluid Mechanics, 2nd edition*. Butterworth Heinemann, 1987.

[30] S. Lanthaler and S. Mishra. On the convergence of the spectral viscosity method for the two-dimensional incompressible euler equations with rough initial data. *Foundations of Computational Mathematics*, 20(5):1309–1362, 10 2020.

[31] S. Lanthaler, S. Mishra, and C. Parés-Pulido. Statistical solutions of the incompressible euler equations. *Mathematical Models and Methods in Applied Sciences*, 31(02):223–292, Feb 2021.

[32] Z. Li, D. Z. Huang, B. Liu, and A. Anandkumar. Fourier neural operator with learned deformations for pdes on general geometries, 2022.

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

[34] Z. Li, N. B. Kovachki, K. Azizzadenesheli, B. Liu, A. M. Stuart, K. Bhattacharya, and A. Anandkumar. Multipole graph neural operator for parametric partial differential equations. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 6755–6766. Curran Associates, Inc., 2020.

[35] Z. Li, K. Meidani, and A. B. Farimani. Transformer for partial differential equations' operator learning, 2023.

[36] P. Lippe and B. S. Veeling. PDE-Refiner: Achieving Accurate Long Rollouts with Neural PDE Solvers.

[37] Z. Liu, H. Hu, Y. Lin, Z. Yao, Z. Xie, Y. Wei, J. Ning, Y. Cao, Z. Zhang, L. Dong, F. Wei, and B. Guo. Swin Transformer V2: Scaling Up Capacity and Resolution, Apr. 2022. arXiv:2111.09883 [cs].

[38] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows, Aug. 2021. arXiv:2103.14030 [cs].

[39] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie. A ConvNet for the 2020s, Mar. 2022. arXiv:2201.03545 [cs].

[40] A. Logg, K.-A. Mardal, and G. N. Wells. *Automated solution of differential equations by the finite element method: The FEniCS book*. Springer, 2012.

[41] I. Loshchilov and F. Hutter. Decoupled weight decay regularization, 2019.

[42] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

[43] F. Luporini, M. Louboutin, M. Lange, N. Kukreja, P. Witte, J. Hückelheim, C. Yount, P. H. J. Kelly, F. J. Herrmann, and G. J. Gorman. Architecture and performance of devito, a system for automated stencil computation. *ACM Trans. Math. Softw.*, 46(1), apr 2020.

[44] K. O. Lye. *Computation of statistical solutions of hyperbolic systems of conservation laws*. PhD thesis, 2020.

[45] K. O. Lye, S. Mishra, and D. Ray. Deep learning observables in computational fluid dynamics. *Journal of Computational Physics*, page 109339, 2020.

[46] K. O. Lye, S. Mishra, D. Ray, and P. Chandrasekhar. Iterative Surrogate Model Optimization (ISMO): An active learning algorithm for PDE constrained optimization with deep neural networks. *Computer Methods in Applied Mechanics and Engineering*, 374:113575, Feb. 2021. arXiv:2008.05730 [cs, math].

[47] A. J. Majda and A. L. Bertozzi. *Vorticity and Incompressible Flow*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2001.

[48] D. A. Masters, N. J. Taylor, T. Rendall, C. B. Allen, and D. J. Poole. Geometric comparison of aerofoil shape parameterization methods. *AIAA Journal*, pages 1575–1589, 2017.

[49] M. McCabe, B. R.-S. Blancard, L. H. Parker, R. Ohana, M. Cranmer, A. Bietti, M. Eickenberg, S. Golkar, G. Krawezik, F. Lanusse, M. Pettee, T. Tesileanu, K. Cho, and S. Ho. Multiple Physics Pretraining for Physical Surrogate Models, Oct. 2023. arXiv:2310.02994 [cs, stat].

[50] S. Mishra, C. Schwab, and J. Šukys. Multi-level Monte Carlo finite volume methods for nonlinear systems of conservation laws in multi-dimensions. *J. Comput. Phys.*, 231(8):3365–3388, 2012.

[51] S. Mishra and A. E. Townsend. *Numerical Analysis meets Machine Learning*. Handbook of Numerical Analysis. Springer, 2024.

[52] D. Mizrahi, R. Bachmann, O. F. Kar, T. Yeo, M. Gao, A. Dehghan, and A. Zamir. 4M: Massively Multimodal Masked Modeling, Dec. 2023. arXiv:2312.06647 [cs].

[53] R. Molinaro, Y. Yang, B. Engquist, and S. Mishra. Neural inverse operators for solving pde inverse problems, 2023.

[54] T. Nguyen, J. Brandstetter, A. Kapoor, J. K. Gupta, and A. Grover. Climax: A foundation model for weather and climate, 2023.

[55] E. Perez, F. Strub, H. de Vries, V. Dumoulin, and A. C. Courville. Film: Visual reasoning with a general conditioning layer. *CoRR*, abs/1709.07871, 2017.

[56] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. W. Battaglia. Learning Mesh-Based Simulation with Graph Networks, June 2021. arXiv:2010.03409 [cs].

[57] M. Prasthofer, T. De Ryck, and S. Mishra. Variable input deep operator networks. *arXiv preprint arXiv:2205.11404*, 2022.

[58] A. Quarteroni, A. Manzoni, and F. Negri. *Reduced basis methods for partial differential equations: an introduction*, volume 92. Springer, 2015.

[59] A. Quarteroni and A. Valli. *Numerical approximation of Partial differential equations*, volume 23. Springer, 1994.

[60] B. Raonić, R. Molinaro, T. De Ryck, T. Rohner, F. Bartolucci, R. Alaifari, S. Mishra, and E. de Bézenac. Convolutional Neural Operators for robust and accurate learning of PDEs, Dec. 2023. arXiv:2302.01178 [cs].

[61] R.Krasny. A study of singularity formation in a vortex sheet with a point vortex approximation. *J. Fluid Mech.*, 167:65–93, 1986.

[62] T. Rohner and S. Mishra. Efficient computation of large-scale statistical solutions to incompressible fluid flows. In *Proceedings of the Platform for Advanced Scientific Computing Conference*, PASC '24. ACM, June 2024.

[63] Y. Rosen, Y. Roohani, A. Agarwal, L. Samotorcan, S. R. Quake, and J. Leskovec. Universal cell embeddings: A foundation model for cell biology. *bioRxiv*, 2023.

[64] K. Saab, T. Tu, W.-H. Weng, R. Tanno, D. Stutz, E. Wulczyn, F. Zhang, T. Strother, C. Park, E. Vedadi, J. Z. Chaves, S.-Y. Hu, M. Schaekermann, A. Kamath, Y. Cheng, D. G. T. Barrett, C. Cheung, B. Mustafa, A. Palepu, D. McDuff, L. Hou, T. Golany, L. Liu, J. baptiste Alayrac, N. Houlsby, N. Tomasev, J. Freyberg, C. Lau, J. Kemp, J. Lai, S. Azizi, K. Kanada, S. Man, K. Kulkarni, R. Sun, S. Shakeri, L. He, B. Caine, A. Webson, N. Latysheva, M. Johnson, P. Mansfield, J. Lu, E. Rivlin, J. Anderson, B. Green, R. Wong, J. Krause, J. Shlens, E. Dominowska, S. M. A. Eslami, K. Chou, C. Cui, O. Vinyals, K. Kavukcuoglu, J. Manyika, J. Dean, D. Hassabis, Y. Matias, D. Webster, J. Barral, G. Corrado, C. Semturs, S. S. Mahdavi, J. Gottweis, A. Karthikesalingam, and V. Natarajan. Capabilities of gemini models in medicine, 2024.

[65] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. Battaglia. Learning to Simulate Complex Physics with Graph Networks. In *Proceedings of the 37th International Conference on Machine Learning*, pages 8459–8468. PMLR, Nov. 2020. ISSN: 2640-3498.

[66] J. Shen, T. Marwah, and A. Talwalkar. Ups: Efficiently building foundation models for pde solving via cross-modal adaptation, 2024.

[67] S. Subramanian, P. Harrington, K. Keutzer, W. Bhimji, D. Morozov, M. Mahoney, and A. Gholami. Towards Foundation Models for Scientific Machine Learning: Characterizing Scaling and Transfer Behavior, May 2023. arXiv:2306.00258 [cs, math].

[68] J. Sun, Y. Liu, Z. Zhang, and H. Schaeffer. Towards a foundation model for partial differential equations: Multi-operator learning and extrapolation. *arXiv preprint arXiv:2404.12355v2*, 2024.

[69] E. Tadmor. Convergence of spectral methods for nonlinear conservation laws. *SIAM Journal on Numerical Analysis*, 26(1):30–44, 1989.

[70] E. Tadmor. Burgers' Equation with Vanishing Hyper-Viscosity. *Communications in Mathematical Sciences*, 2(2):317 – 324, 2004.

[71] M. Takamoto, T. Praditia, R. Leiteritz, D. MacKinlay, F. Alesiani, D. Pflüger, and M. Niepert. PDEBENCH: An Extensive Benchmark for Scientific Machine Learning, Mar. 2023. arXiv:2210.07182 [physics].

[72] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample. Llama: Open and efficient foundation language models, 2023.

[73] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush. Transformers: State-of-the-Art Natural Language Processing. pages 38–45. Association for Computational Linguistics, Oct. 2020.

[74] L. Yang, S. Liu, T. Meng, and S. J. Osher. In-context operator learning with data prompts for differential equation problems. *Proceedings of the National Academy of Sciences*, 120(39):e2310142120, Sept. 2023. Publisher: Proceedings of the National Academy of Sciences.

[75] Y. Zhu and N. Zabaras. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 336:415–447, 2018.

Supplementary Material for:
POSEIDON: Efficient Foundation Models for PDEs.

Table of Contents

A Architecture of the scalable Operator Transformer (scOT)

A.1 Operator Learning with scOT

First, we describe how scOT (Section 2 of Main Text and Figure 2) transforms function space inputs into function outputs below.

For simplicity of exposition, we set $d = 2$ and specify $D = [0, 1]^2$ as the underlying domain. On this domain, an uniform *computational grid*, with grid spacing Δ, of J^2 equally spaced points $x_{j_x,j_y} = (j_x\Delta, j_y\Delta)$, with $J = 1/\Delta$, is set. Let $1 < p < J$ such that $J \mathrm{mod} p = 0$ and set $P = J/p$. We divide the domain $D = \cup_{\rho=1}^{P^2} D_\rho$ into a set of P^2 non-overlapping and equal (in measure) *patches*. Any underlying input function $a \in C(D; \mathbb{R}^n)$ is then *partitioned* into a function, that is piecewise constant on patches and *embedded* into a C-dimensional latent representation by applying the operator,

$$\mathbf{v}(x) = \hat{\mathbf{E}}(a)(x) = \sum_{\rho=1}^{J^2} \mathbf{F} \left(\int_{D_\rho} W(x)a(x)dx \right) \mathbb{I}_{D_\rho}(x), \tag{12}$$

with $\mathbf{F} \in \mathbb{R}^{C\times n}$ is a learnable matrix and the weight function W is defined in terms of the underlying computational grid as, $W(x) = \sum\limits_{1\leq j_x,j_y\leq J} W_{ij}\delta_{x_{j_x j_y}}$, with δ denoting the Dirac measure, and the shared (across all patches) learnable weights given by,

$$W_{j_x j_y} = \begin{cases} \omega_{j_x j_y} & \text{if} \quad 1 \leq j_x, j_y \leq p \\ \omega_{j_x \mathrm{mod} p, j_y \mathrm{mod} p}, & \mathrm{otherwise}. \end{cases} \tag{13}$$

The (patched and embedded) output function $\mathbf{v}$ of (12) is then processed through a sequence of *SwinV2 transformer* blocks [38, 37], each of which has the structure of $SW_\ell : C(D; \mathbb{R}^C) \mapsto C(D; \mathbb{R}^C)$, for layer index $\ell = 1, \cdots, L$, formulated in Main Text (3).

The main building block of a SwinV2 transformer block (3) is the *windowed multi-head self attention* operator,

$$W - MSA^\ell(\mathbf{v})(x) = \sum_{h=1}^{H} \mathbf{W}_\ell^h \int_{D^\ell_{q_x}} \frac{e^{\left(\cos\left(\mathbf{Q}_\ell^h\mathbf{v}(x),\mathbf{K}_\ell^h\mathbf{v}(y)\right)+\mathbf{B}_\ell^h(x,y)\right)}}{\int_{D^\ell_{q_x}} e^{\left(\cos\left(\mathbf{Q}_\ell^h\mathbf{v}(z),\mathbf{K}_\ell^h\mathbf{v}(y)\right)+\mathbf{B}_\ell^h(z,y)\right)}dz} \mathbf{V}_\ell^h \mathbf{v}(y)dy, \tag{14}$$

for any $\mathbf{v} \in C(D; \mathbb{R}^C)$. Here, h denotes the h-th attention head, $\mathbf{W}_\ell^h \in \mathbb{R}^{C\times m}$ be the output matrix and $\mathbf{Q}_\ell^h, \mathbf{K}_\ell^h, \mathbf{V}_\ell^h \in \mathbb{R}^{m\times C}$ be the *query, key* and *value* matrices, respectively. For any two vectors α, β, the cosine similarity is defined as $\langle\alpha, \beta\rangle = |\alpha||\beta| \cos(\alpha, \beta)$ and $\mathbf{B}_\ell^h : D \times D \mapsto \mathbb{R}$ is a general form for *positional encodings*. To be more specific, we use *relative log position encodings* by setting the inputs to $\mathbf{B}_\ell^h$ to be the logarithm of the relative positions $(k, \bar{k})$ within the window and the function $\mathbf{B}_\ell^h$ itself to be a small MLP. Finally, the domain of integration $D^\ell_{q_x}$ is simply the window where the point of interest x lies, i.e., $1 \leq q_x \leq M^2$ such that $x \in D^\ell_{q_x}$. Underlying (14), is the partition of the domain into windows such that $D = \cup_{q=1}^{M^2} D^\ell_q$, with $1 \leq \ell \leq L$ indexing the underlying layer within a SwinV2 transformer block and with M^2, denoting the number of windows. Moreover, the windows are shifted across layers, as depicted in Figure 2 (c), so that all the points can be attended to, by iteratively shifting windows across multiple layers/blocks.

The MLP in Main Text (3) is of the form, $MLP : C(D; \mathbb{R}^C) \mapsto C(D; \mathbb{R}^C)$ with

$$MLP(\mathbf{v})(x) = \bar{W}\sigma\left(W\mathbf{v}(x) + \hat{B}\right), \tag{15}$$

for learnable weight matrices $W \in \mathbb{R}^{\bar{C}\times C}$, $\bar{W} \in \mathbb{R}^{C\times\bar{C}}$, bias vector $\hat{B} \in \mathbb{R}^{\bar{C}}$ and nonlinear activation function $\sigma : \mathbb{R} \mapsto \mathbb{R}$. The Layer Norm LN in Main Text (3) is given by Main Text (4). The remaining operations in scOT (see Main Text Figure 2) are described in their discrete form below.

A.2 Computational Realization of scOT

The scalable Operator Transformer (scOT), forming the underlying model architecture for POSEIDON, is constructed as an encoder-decoder architecture. Starting from patching and embedding, embedded tokens are inputted into multiple stages of SwinV2 transformer blocks, each followed by a patch merging. The encoder is connected at every level to the decoder through ConvNeXt [39] blocks, whereas the bottleneck is convolution-free. Finally, through patch recovery and mixup, the output is assembled. We refer to Figure 2 (a) in the Main Text for an illustration of the overall architecture and concrete computational realizations by presenting discrete versions of the continuous operators described in the subsection above as well as elaborating on other operators used in scOT.

Patch Partitioning. The encoder consists of the patch partitioning operation, creating visual tokens from n discretized (on the uniform computational grid described in Section A.1) input functions $\mathbf{a}_i \in \mathbb{R}^{J\times J}$, $i \in \{1, ..., n\}$. Each $\mathbf{a}_i$ is divided into non-overlapping patches of size $p \times p$ (with $p \ll J$) such that $P^2 = \left\lceil \frac{J}{p} \right\rceil^2$ patches arise. For an illustration, we refer to Figure 2 (c) of the Main Text where $P = 8$. Patches are combined for every a_i such that a sequence of $\mathbf{a}_j^p \in \mathbb{R}^{p\times p\times n}$, $j \in \{1, ..., P^2\}$ visual tokens can be fed to the embedding operation.

Embedding. Each of these patches is *linearly* embedded using a shared learnable weight $\mathbf{W}_\mathcal{E} \in \mathbb{R}^{C\times n\times p\times p}$ ($\in \tilde{\Theta}$) and bias $\mathbf{b}_\mathcal{E} \in \mathbb{R}^C$ ($\in \tilde{\Theta}$),

$$(\mathbf{v}_j)_i = (\mathbf{b}_\mathcal{E})_i + \sum_{k=1}^{c} \sum_{u,v=1}^{p} (\mathbf{W}_\mathcal{E})_{i,k,u,v} (\mathbf{a}_j^p)_{k,u,v} \tag{16}$$

where $(\cdot)_i$ denotes the i-th component (for all $1 \leq i \leq C$) and $C > n$ is the embedding dimension. It is straightforward to observe that (16) is a discretization of the operator (12), with an additional bias term. The resulting embedding is then passed through a (time-conditioned) layer norm (see Main Text Equation 23).

SwinV2 Stage. At each level $i \in \{0, ..., L-1\}$ of the U-Net-style architecture, a SwinV2 stage $\mathcal{S}_i$ is employed consisting of t_i chained SwinV2 transformer blocks $\mathcal{T}_{t_i}$,

$$\mathcal{S}_i = \mathcal{T}_{t_i} \circ \mathcal{T}_{t_i - 1} \circ ... \circ \mathcal{T}_1. \tag{17}$$

This is done in both encoder and decoder, and the same number t_i of SwinV2 transformer blocks is used on each level.

SwinV2 Transformer Block. A SwinV2 transformer block $\mathcal{T}$ is built as follows

$$\mathbf{v}'(\mathbf{v}) = (\mathcal{N} \circ \mathcal{A})(\mathbf{v}) + \mathbf{v} \tag{18}$$

$$\mathcal{T}(\mathbf{v}) = (\mathcal{N} \circ \mathcal{M})(\mathbf{v}'(\mathbf{v})) + \mathbf{v}'(\mathbf{v}) \tag{19}$$

where $\mathbf{v} \in \mathbb{R}^{P^2/4^i \times C\cdot 2^i}$ is the sequence of embedded tokens, $\mathcal{A}$ the shifted-window multi-head self-attention operation, $\mathcal{N}$ the (time-conditioned) Layer Norm, $\mathcal{M}$ a MLP. The attention mechanism $\mathcal{A}$ acts only on windows of size $M \times M$ patches/tokens that shift from block $\mathcal{T}_l$ to block $\mathcal{T}_{l+1}$ by doing a cyclic displacement of $M/2 \times M/2$ tokens (when the sequence is interpreted in 2D; see Figure 2 (c) of Main Text). So, with an input window $\mathbf{v} \in \mathbb{R}^{M^2 \times C\cdot 2^i}$,

$$\mathcal{A}(\mathbf{v}) = \text{Concat}[\mathcal{A}_1(\mathbf{v}), ..., \mathcal{A}_{h_i}(\mathbf{v})]\mathbf{W}_O + \mathbf{b}_O^\top \mathbb{I} \tag{20}$$

where $\mathcal{A}_l$, $1 \leq l \leq h_i$ is attention in head l with the maximum number of heads depending on the stage i, with $\mathbf{W}_O \in \mathbb{R}^{C\cdot 2^i \times C\cdot 2^i}$, $\mathbf{b}_O \in \mathbb{R}^{C\cdot 2^i}$ being learnable parameters ($\in \hat{\Theta}$). $\mathcal{A}_l$ is then given by

$$\mathcal{A}_l(\mathbf{v}) = \text{Softmax}\left(\mathbf{B}_l(\mathbf{v}) + \frac{\cos((\mathbf{v}\mathbf{W}_Q^l + \mathbf{1}_{M^2} \cdot \mathbf{b}_Q^{l\top})^\top, (\mathbf{v}\mathbf{W}_K^l)^\top)}{\tau_l}\right) \cdot \left(\mathbf{v}\mathbf{W}_V^l + \mathbf{1}_{M^2} \cdot \mathbf{b}_V^{l\top}\right) \tag{21}$$

with $\mathbf{W}_V^l, \mathbf{W}_Q^l, \mathbf{W}_K^l \in \mathbb{R}^{C\cdot 2^i \times C\cdot 2^i/h_i}$ and $\mathbf{b}_Q^l, \mathbf{b}_V^l \in \mathbb{R}^{C\cdot 2^i/h_i}$, $\tau_l \in \mathbb{R}$ (all learnable $\in \hat{\Theta}$), $\cos(\cdot, \cdot)$ the cosine similarity, $\mathbf{1}_{M^2} \in \mathbb{R}^{M^2}$ a vector of ones, $\mathbf{B}_l(\mathbf{v}) \in \mathbb{R}^{M^2 \times M^2}$ the relative position bias matrix generated from the (logarithmic) relative positions of each patch $[\Delta x, \Delta y]^\top$ within a window, parametrized through a shared MLP $\mathcal{P}$ for all heads:

$$\mathcal{P}(\Delta x, \Delta y) = \text{ReLU}([\text{sign}(\Delta x)\log(1+|\Delta x|), \text{sign}(\Delta y)\log(1+|\Delta y|)]^\top \mathbf{W}_{B,1} + \mathbf{b}_{b,1})\mathbf{W}_{B,2} \tag{22}$$

$\mathbf{W}_{B,1} \in \mathbb{R}^{2\times 512}$, $\mathbf{b}_{b,1} \in \mathbb{R}^{512}$, $\mathbf{W}_{B,2} \in \mathbb{R}^{512\times h_i}$ are all learnable ($\in \hat{\Theta}$). Note that (21) is a discretization of the operator (14) by replacing the spatial integrals therein with uniform quadrature.

The tokens, coming from the attention module are then fed to a layer norm [2] if the PDE to be learned is time-independent; if it is time-dependent (also in the case of POSEIDON), it goes through a time-conditioned layer norm [55] $\mathcal{N}$

$$\mu(\mathbf{v}) = \frac{1}{C\cdot 2^i}\sum_{l=1}^{C\cdot 2^i}(\mathbf{v})_l \tag{23}$$

$$\sigma^2(\mathbf{v}) = \frac{1}{C\cdot 2^i}\sum_{l=1}^{C\cdot 2^i}((\mathbf{v})_l - \mu(\mathbf{v}))^2 \tag{24}$$

$$\mathcal{N}(\mathbf{v}, t) = \alpha(t) \odot \frac{\mathbf{v} - \mu(\mathbf{v})\cdot \mathbf{1}_{C\cdot 2^i}}{\sigma^2(\mathbf{v})} + \beta(t) \tag{25}$$

with $\mathbf{v} \in \mathbb{R}^{C\cdot 2^i}$ be a token resulting from the attention module, $t \in \mathbb{R}_{\geq 0}$, $\mathbf{1}_{C\cdot 2^i} \in \mathbb{R}^{C\cdot 2^i}$ a vector of ones, and $\alpha(t) = \mathbf{W}_\alpha t + \mathbf{b}_\alpha$, $\beta(t) = \mathbf{W}_\beta t + \mathbf{b}_\beta$ being learnable gain and bias ($\mathbf{W}_\alpha, \mathbf{W}_\beta, \mathbf{b}_\alpha, \mathbf{b}_\beta \in \mathbb{R}^{C\cdot 2^i}$, all $\in \tilde{\Theta}'$).

The last building block of the SwinV2 transformer block is a single-hidden-layer MLP with GeLU [21] as pointwise activation function and four times the width of the latent dimension $C\cdot 2^i$

$$\mathcal{M}(\mathbf{v}) = \mathrm{GeLU}(\mathbf{v}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2 \tag{26}$$

with $\mathbf{W}_1 \in \mathbb{R}^{C\cdot 2^i \times 4\cdot C\cdot 2^i}$, $\mathbf{b}_1 \in \mathbb{R}^{4\cdot C\cdot 2^i}$, $\mathbf{W}_2 \in \mathbb{R}^{4\cdot C\cdot 2^i \times C\cdot 2^i}$, $\mathbf{b}_2 \in \mathbb{R}^{C\cdot 2^i}$ all learnable parameters ($\in \hat{\Theta}$).

Patch Merging. At each (resolution) level i of the architecture, after each SwinV2 stage in the encoder, a *linear* downsampling operation $\mathcal{D}_i$ is performed on the output of the stage added to its input (additional residual connection) such that the resolution halves. This amounts to a linear transformation on four non-overlapping, stacked patches/tokens at a time $\mathbf{v} \in \mathbb{R}^{4\cdot C\cdot 2^i}$

$$\mathcal{D}_i(\mathbf{v}, t) = \mathcal{N}(\mathbf{W}_{\mathcal{D}_i}\mathbf{v}, t) \tag{27}$$

with learnable $\mathbf{W}_{D_i} \in \mathbb{R}^{C\cdot 2^{i+1}\times 4\cdot C\cdot 2^i}$ such that the latent dimension doubles. Here, an additional (time-conditioned) layer norm is applied.

ConvNeXt Block. Outputs from each encoder stage $\mathcal{S}_i$, $0 \leq i \leq L-2$ are fed to n_c chained (time-conditioned) ConvNeXt blocks [39] $\mathcal{Q}_i$; for that, the token sequence is reshaped to a two-dimensional grid of tokens $\mathbf{v} \in \mathbb{R}^{P/2^i \times P/2^i \times C\cdot 2^i}$ and transformed by

$$\mathcal{Q}_i(\mathbf{v}, t) = (\mathrm{GeLU}(\mathcal{N}(\mathrm{DwConv}(\mathbf{v}), t)\mathbf{W}_{\mathcal{Q},1} + \mathbf{b}_{\mathcal{Q},1})\mathbf{W}_{\mathcal{Q},2} + \mathbf{b}_{\mathcal{Q},2}) \odot \mathbf{W}_{\mathcal{Q},3} + \mathbf{v} \tag{28}$$

$\mathbf{W}_{\mathcal{Q},1} \in \mathbb{R}^{C\cdot 2^i \times 4\cdot C\cdot 2^i}$, $\mathbf{b}_{\mathcal{Q},1} \in \mathbb{R}^{4\cdot C\cdot 2^i}$, $\mathbf{W}_{\mathcal{Q},2} \in \mathbb{R}^{4\cdot C\cdot 2^i \times C\cdot 2^i}$, $\mathbf{b}_{\mathcal{Q},2} \in \mathbb{R}^{C\cdot 2^i}$, $\mathbf{W}_{\mathcal{Q},3} \in \mathbb{R}^{C\cdot 2^i}$ all learnable parameters ($\in \hat{\Theta}$) and DwConv is a depthwise convolution with kernel size 7 (and a padding of 3) and bias.

Patch Expansion. Similar to patch merging, after a SwinV2 stage in the decoder, each output token $\mathbf{v} \in \mathbb{R}^{C\cdot 2^{i+1}}$ is *linearly* upsampled through $\mathcal{U}_i$ to double the resolution and half the latent dimension,

$$\mathcal{U}_i(\mathbf{v}, t) = \mathcal{N}(\mathrm{Reshape}(\mathbf{W}_{\mathcal{U}_{i,1}}\mathbf{v}), t)\mathbf{W}_{\mathcal{U}_{i,2}} \tag{29}$$

where $\mathbf{W}_{\mathcal{U}_{i,1}} \in \mathbb{R}^{C\cdot 2^{i+2}\times C\cdot 2^{i+1}}$, $\mathbf{W}_{\mathcal{U}_{i,2}} \in \mathbb{R}^{C\cdot 2^i \times C\cdot 2^i}$ are both learnable ($\in \hat{\Theta}$), and Reshape$(\cdot)$ an operation that reshapes a vector of size $C\cdot 2^{i+2}$ into a matrix of size $4 \times C\cdot 2^i$.

Patch Recovery and Mixup. Having passed through the last stage of the decoder, every patch/visual token $\mathbf{v}_j \in \mathbb{R}^C$ is *linearly* transformed back from the latent space to form patches of the discretized output function $\mathbf{u}_j^p \in \mathbb{R}^{p\times p\times c_u}$,

$$(\mathbf{u}_j^p)_i = (\mathbf{b}_{\mathcal{R}})_i \mathbb{I} + \sum_{k=1}^{C}(\mathbf{W}_{\mathcal{R}})_{i,k,*,*}(\mathbf{v}_j)_k \tag{30}$$

where $(\cdot)_i$ denotes the i-th component (for all $1 \leq i \leq c_u$) and c_u is the number of components of the discretized output function. $\mathbf{W}_{\mathcal{R}} \in \mathbb{R}^{c_u \times C \times p \times p}$ and $\mathbf{b}_{\mathcal{R}} \in \mathbb{R}^{c_u}$ are shared across tokens and learnable ($\in \tilde{\Theta}$). These outputs are assembled on a grid to form $\tilde{\mathbf{u}} \in \mathbb{R}^{J \times J \times c_u}$ which is transformed to the final output $\mathbf{u}$ with a convolution with kernel size 5 (and padding 2 to keep the dimensionality), without bias, with all parameters being in $\tilde{\Theta}$.

Summary of Hyperparameters. In Table 2, we give an overview of the hyperparameters to instantiate a scOT. To reduce the number of hyperparameters, we fix $p = 4$, $M = 16$, $L = 4$, $[h_1, h_2, h_3, h_4] = [3, 6, 12, 24]$, and $n_c = 2$ in this work.

Table 2: Hyperparameters of scOT.

Hyperparameter	Description
p	patch size
M	window size
C	embedding/latent dimension
L	number of levels ($L - 1$ downsampling/upsampling operations)
t_i	number of SwinV2 transformer blocks in level i
h_i	number of attention heads in level i
n_c	number of ConvNeXt blocks at each level

B Datasets

We describe the various datasets used for pretraining and for the downstream tasks below. All these datasets are publicly available with the PDEGym collection (https://huggingface.co/collections/camlab-ethz/pdegym-665472c2b1181f7d10b40651).

B.1 Pretraining Datasets

Table 3: Abbreviations/Summary for all the pretraining datasets. IC refers to initial conditions.

Abbreviation	PDE	Defining Feature	Visualization
NS-Sines	Navier-Stokes (31)	Sine IC	Fig. 55
NS-Gaussians	Navier-Stokes (31)	Gaussians (in Vorticity) IC	Fig. 56
CE-RP	Euler (37)	4-Quadrant Riemann Problem IC	Fig. 57
CE-CRP	Euler (37)	Multiple Curved Riemann Problems	Fig. 58
CE-KH	Euler (37)	Shear IC	Fig. 59
CE-Gauss	Euler (37)	Gaussians (in Vorticity) IC	Fig. 60

We pretrain Poseidon models and CNO-FM on a dataset containing 6 operators, defined on the space-time domain $[0,1]^2 \times [0,1]$. We include 2 operators governed by the Navier-Stokes equations (NS-Sines, NS-Gauss) and 4 operators governed by the Compressible Euler equations. The pretraining datasets encompass problems that exhibit a wide range of scales and complex, nonlinear dynamics.

The *Incompressible Navier-Stokes equations* of fluid dynamics are given by

$$u_t + (u \cdot \nabla)u + \nabla p = \nu \Delta u, \quad \text{div } u = 0, \tag{31}$$

in the domain $D = [0,1]^2$ with suitable boundary conditions. Here, $u : [0,T] \times D \mapsto \mathbb{R}^2$ is the velocity field and $p : [0,T] \times D \mapsto \mathbb{R}_+$ is the pressure. In this work, a small viscosity $\nu = 4 \times 10^{-4}$ is only applied to high-enough Fourier modes to approximate the inviscid limit.

To generate the pretraining data, all the benchmarks for the Navier-Stokes equations are simulated until the time $T = 1$. Furthermore, we store *21 snapshots* of the numerically simulated velocity field u, uniformly spaced in time. Each snapshot has a spatial resolution of 128×128. The initial conditions are drawn from various distributions, which we will describe later. The distribution of these initial conditions is crucial for determining the complexity of the samples and the overall dynamics.

All the Navier-Stokes experiments, both for the pretraining dataset and the downstream tasks, are simulated with the following *spectral method*. Fix a mesh width $\Delta = \frac{1}{N}$ for some $N \in \mathbb{N}$. We consider the following discretization of the Navier-Stokes equations in the Fourier domain

$$\begin{cases} \partial_t u^\Delta + \mathcal{P}_N(u^\Delta \cdot \nabla u^\Delta) + \nabla p^\Delta & = \varepsilon_N \Delta(Q_N * u^\Delta) \\ \nabla \cdot u^\Delta & = 0 \\ u^\Delta|_{t=0} & = \mathcal{P}_N u_0 \end{cases} \tag{32}$$

where $\mathcal{P}_N$ is the spatial Fourier projection operator mapping a function $f(x,t)$ to its first N Fourier modes: $\mathcal{P}_N = \sum_{|k|_\infty \le N} \hat{f}_k(t) e^{ik\cdot x}$. The artificial viscosity term we use for the stabilization of the solver consists of a resolution-dependent viscosity ε_N and a Fourier multiplier Q_N controlling the strength at which different Fourier modes are dampened. This allows us to not dampen the low frequency modes, while applying some diffusion to the problematic higher frequencies. The Fourier multiplier Q_N is of the form

$$Q_N(\mathbf{x}) = \sum_{\mathbf{k} \in \mathbb{Z}^d, |\mathbf{k}| \le N} \hat{Q}_{\mathbf{k}} e^{i\mathbf{k}\cdot\mathbf{x}}. \tag{33}$$

In order for the solver to converge, the Fourier coefficients of Q_N need to fulfill [69, 70, 31]

$$\hat{Q}_k = 0 \text{ for } |k| \le m_N, 1 - \left(\frac{m_N}{|k|}\right)^{\frac{1}{\theta}} \le \hat{Q}_k \le 1 \tag{34}$$

where we have introduced an additional parameter $\theta > 0$. The quantities m_N and ε_N are required to scale as

$$m_N \sim N^\theta, \varepsilon_N \sim \frac{1}{N}, 0 < \theta < \frac{1}{2}. \tag{35}$$

For the experiment described here, we choose $m_N = \sqrt{N}$, $\varepsilon_N = \frac{0.05}{N}$, and $N = 128$. This gives rise to the viscosity $\nu \approx 4 \cdot 10^{-4}$ mentioned above. The Fourier multipliers $\hat{Q}_N$ are chosen according to [30] and are equal to

$$\hat{Q}_{\mathbf{k}}^{\text{Smooth}} = 1 - \exp\left(-\left(\frac{|\mathbf{k}|}{k_0}\right)^\alpha\right). \tag{36}$$

The Navier-Stokes simulations were performed with the *AZEBAN* spectral hyperviscosity code [62].

The Compressible Euler equations of gas dynamics are given by

$$u_t + \operatorname{div} F(u) = 0,\ u = [\rho, \rho v, E]^\perp,\ F = [\rho v, \rho v \otimes v + p\mathbf{I}, (E + p)]v]^\perp, \tag{37}$$

in the domain $[0, 1]^2$ with suitable boundary conditions, with density ρ, velocity $v = [v_x, v_y]$, pressure p and total Energy E related by the ideal gas equation of state:

$$E = \frac{1}{2}\rho|u|^2 + \frac{p}{\gamma - 1}, \tag{38}$$

where $\gamma = 1.4$. All the trajectories are simulated until time $T = 1$. The simulations for the compressible Euler equations were performed with the *ALSVINN* [44] code, which is based on a high-resolution finite volume scheme with piecewise quadratic WENO reconstructions and HLLC Riemann solvers.

During pretraining, our goal is to predict four variables: $[\rho, v_x, v_y, p]$, where ρ represents density, v_x is the horizontal velocity, v_y is the vertical velocity, and p is the pressure. As in the Navier-Stokes benchmarks, all the trajectories for compressible Euler are simulated until time $T = 1$. Furthermore, we store *21 snapshots* of the numerically simulated solution, uniformly spaced in time. Each snapshot has a spatial resolution of 128×128, though being generated on 512×512 and downsampled.

Next we describe each constituent of the pretraining dataset (summarized in Table 3)

B.1.1 NS-Sines

This dataset considers the incompressible Navier-Stokes equations (31) with the following initial conditions,

$$\begin{aligned} u_x^0(x, y) &= \sum_{i,j=1}^{p} \frac{\alpha_{i,j}}{\sqrt{2\pi(i + j)}} \sin(2\pi i x + \beta_{i,j}) \sin(2\pi j y + \gamma_{i,j}) \\ u_y^0(x, y) &= \sum_{i,j=1}^{p} \frac{\alpha_{i,j}}{\sqrt{2\pi(i + j)}} \cos(2\pi i x + \beta_{i,j}) \cos(2\pi j y + \gamma_{i,j}) \end{aligned} \tag{39}$$

where the random variables are chosen as $\alpha_{i,j} \sim \mathcal{U}_{[-1,1]}$, $\beta_{i,j} \sim \mathcal{U}_{[0,2\pi]}$, and $\gamma_{i,j} \sim \mathcal{U}_{[0,2\pi]}$. The number of modes p is chosen to be $p = 10$. Thus, the initial conditions amount to a linear combination of sines and cosines.

The underlying solution operator $\mathcal{S}(t, \cdot)$ is given by $\mathcal{S}(t, u_{x,y}^0) = u_{x,y}(t)$, with u_x, u_y solving the Navier-Stokes equations (31) with periodic boundary conditions.

We generated 20000 NS-Sines trajectories of which the first 19640 belong to the training set, the next 120 to the validation set, and the last 240 to the test set. Note that we included 11 time steps in the pretraining dataset, with every other time step selected, starting from step 0 up to step 21. A visualization of a random sample and the predictions made by POSEIDON-B (trained on 128 training trajectories) is shown in Figure 55.

B.1.2 NS-Gauss

Given a two-dimensional velocity field $u = (u_x, u_y)$, its *vorticity* is given by the scalar $\omega = \operatorname{curl} u = \partial_x u_y - \partial_y u_x$. Note that, for any time t, the velocity can be recovered from the vorticity using the so-called *stream function* [47].

For this dataset, we specify the initial conditions for the Navier-Stokes equations in terms of the vorticity given by,

$$\omega_0(x, y) = \sum_{i=1}^{p} \frac{\alpha_i}{\sigma_i} \exp\left(-\frac{(x - x_i)^2 + (y - y_i)^2}{2\sigma_i^2}\right) \tag{40}$$

where we chose $p = 100$ Gaussians with $\alpha_i \sim \mathcal{U}_{[-1,1]}$, $\sigma_i \sim \mathcal{U}_{[0.01,0.1]}$, $x_i \sim \mathcal{U}_{[0,1]}$, and $y_i \sim \mathcal{U}_{[0,1]}$. Thus, the initial vorticity field is a superposition of a large number of Gaussians. The initial velocity field is then recovered from the vorticity.

The underlying solution operator $\mathcal{S}(t, \cdot)$ is given by $\mathcal{S}(t, u^0_{x,y}) = u_{x,y}(t)$, with u_x, u_y solving the Navier-Stokes equations (31) with periodic boundary conditions.

We generated 20000 NS-Gauss trajectories with the same train/validation/test split and time-stepping as for NS-Sines. A visualization of a random sample and the predictions made by POSEIDON (128 training trajectories) are shown in Figure 56.

B.1.3 CE-RP

The well-known four-quadrant Riemann problem is the generalization of the standard Sod shock tube to two-space dimensions [22]. It is defined by dividing the domain $D = [0, 1]^2$ into a grid of $p \times p$ square subdomains

$$D_{i,j} = \left\{(x, y) \in \mathbb{T}^2 \mid \frac{i-1}{p} \leq x < \frac{i}{p}, \frac{j-1}{p} \leq y < \frac{j}{p}\right\},$$

where $\mathbb{T}^2$ is the 2d torus. We fix $p = 2$ for this problem.

The initial data on each of these subdomains is constant and given by,

$$(\rho_0, v^0_x, v^0_y, p_0) = (\rho_{i,j}, (v_x)_{i,j}, (v_y)_{i,j}, p_{i,j}).$$

By sampling $\rho_{i,j} \sim \mathcal{U}_{[0.1,1]}$, $(v_x)_{i,j} \sim \mathcal{U}_{[-1,1]}$, $(v_y)_{i,j} \sim \mathcal{U}_{[-1,1]}$, and $p_{i,j} \sim \mathcal{U}_{[0.1,1]}$, we obtain a stochastic version of the four-quadrant Riemann problem, which also generalizes the stochastic shock tubes of [50] to two-space dimensions.

The underlying solution operator $\mathcal{S}(t, \cdot)$ is given by $\mathcal{S}(t, \rho_0, v^0_{x,y}, p_0) = [\rho(t), v_{x,y}(t), p(t)]$ solving the compressible Euler equations (37) with periodic boundary conditions.

We generated 10000 CE-RP trajectories where the first 9640 trajectories belong to the training set, the following 120 to the validation set, and the last 240 trajectories to the test set. The time-stepping is the same as for NS-Sines and NS-Gauss. A visualization of a random sample and the predictions made by POSEIDON (128 finetuning trajectories) are shown in Figure 57.

B.1.4 CE-CRP

This dataset corresponds to a *curved* and multi-partitioned version of the CE-RP dataset. To define it, we denote the fractional part of $x \in \mathbb{R}$ as $\{x\} := x - \lfloor |x| \rfloor \operatorname{sgn} x$ and define the functions

$$\sigma_x(x, y) = \sum_{i,j=1}^{p} \alpha_{x,i,j} \sin(2\pi i x + j y + \beta_{x,i,j})$$

$$\sigma_y(x, y) = \sum_{i,j=1}^{p} \alpha_{y,i,j} \sin(2\pi i x + j y + \beta_{y,i,j}).$$

where $\alpha_{k,i,j} \sim \mathcal{U}_{[-0.1,0.1]}$, and $\beta_{k,i,j} \sim \mathcal{U}_{[0,1]}$. These functions are then used to create a partition of the domain into curved subdomains,

$$D_{i,j} = \{(x, y) \in \mathbb{T}^2 \mid x_{\text{min}} \leq \{x + \sigma_x(x, y) + 1\} < x_{\text{max}}, y_{\text{min}} \leq \{y + \sigma_y(x, y) + 1\} < y_{\text{max}}\}.$$

with $x_{\text{min}} = \frac{i}{p+1}$, $x_{\text{max}} = \frac{i+1}{p+1}$, $y_{\text{min}} = \frac{j}{p+1}$, and $y_{\text{max}} = \frac{j+1}{p+1}$. Finally, the initial conditions are given by

$$(\rho, v_x, v_y, p)|_{t=0} = (\rho_{i,j}, u_{i,j}, v_{i,j}, p_{i,j}) \text{ in } D_{i,j}$$

where $\rho_{i,j} \sim \mathcal{U}_{[0.1,1]}$, $(v_x)_{i,j} \sim \mathcal{U}_{[-1,1]}$, $(v_y)_{i,j} \sim \mathcal{U}_{[-1,1]}$, and $p_{i,j} \sim \mathcal{U}_{[0.1,1]}$. A visualization of a random sample of the initial conditions is shown in Figure 58 and illustrates how this problem is a curved, multi-partitioned version of the standard stochastic four-quadrant Riemann problem (CE-RP).

The underlying solution operator $\mathcal{S}(t, \cdot)$ is given by $\mathcal{S}(t, \rho_0, v^0_{x,y}, p_0) = [\rho(t), v_{x,y}(t), p(t)]$ solving the compressible Euler equations (37) with periodic boundary conditions.

We generated 10000 CE-CRP trajectories with the same train/validation/test split as CE-RP. The time-stepping is the same as for NS-Sines and NS-Gauss. A visualization of a random sample and the predictions made by POSEIDON (128 training trajectories) are shown in Figure 58.

B.1.5 CE-KH

This is a well-known benchmark of compressible fluid dynamics that corresponds to the well-known Kelvin-Helmholtz instability [29]. A modern version is presented, for instance, in [16].

The underlying initial data is,

$$(\rho, v_x, v_y, p)|_{t=0} = \begin{cases} (1, 0.5, 0, 2.5) & \text{if } y < 0.25 + \sigma_0(x) \text{ or } y > 0.75 + \sigma_1(x) \\ (2, -0.5, 0, 2.5) & \text{otherwise.} \end{cases}$$

The perturbations σ_0 and σ_1 are given by

$$\sigma_i(x) = \frac{\varepsilon}{\sum_{j=1}^{p} \alpha_{i,j}} \sum_{j=1}^{p} \alpha_{i,j} \cos(2\pi j(x + \beta_{i,j}))$$

where $\varepsilon = 0.05$, $\alpha_{i,j} \sim \mathcal{U}_{[0,1]}$, and $\beta_{i,j} \sim \mathcal{U}_{[0,1]}$.

The underlying solution operator $\mathcal{S}(t, \cdot)$ is given by $\mathcal{S}(t, \rho_0, v^0_{x,y}, p_0) = [\rho(t), v_{x,y}(t), p(t)]$ solving the compressible Euler equations (37) with periodic boundary conditions.

We generated 10000 CE-KH trajectories with the same train/validation/test split as CE-RP. The time-stepping is the same as for NS-Sines and NS-Gauss. A visualization of a random sample and the predictions made by POSEIDON (128 training trajectories) are shown in Figure 59.

B.1.6 CE-Gauss

As in the NS-Sines dataset, we initialize the curl ω of the initial velocity with a superposition of Gaussians,

$$\omega_0(x, y) = \sum_{i=1}^{p} \frac{\alpha_i}{\sigma_i} \exp\left(-\frac{(x - x_i)^2 + (y - y_i)^2}{2\sigma_i^2}\right)$$

where we chose $p = 100$ Gaussians with $\alpha_i \sim \mathcal{U}_{[-1,1]}$, $\sigma_i \sim \mathcal{U}_{[0.01,0.1]}$, $x_i \sim \mathcal{U}_{[0,1]}$, and $y_i \sim \mathcal{U}_{[0,1]}$. Then, the initial field is generated from the vorticity by using the incompressibility condition. The underlying density and pressure are initialized with constants, $\rho = 0.1$ and $p = 2.5$, respectively.

The underlying solution operator $\mathcal{S}(t, \cdot)$ is given by $\mathcal{S}(t, \rho_0, v^0_{x,y}, p_0) = [\rho(t), v_{x,y}(t), p(t)]$ solving the compressible Euler equations (37) with periodic boundary conditions.

We generated 10000 CE-Gauss trajectories with the same train/validation/test split as CE-RP. Time-stepping is the same as for NS-Sines and NS-Gauss. A visualization of a random sample and the predictions made by POSEIDON (128 training trajectories) are shown in Figure 60.

We remark that out of the 6 operators that consitute the pretraining dataset, 2 of them (CE-KH and CE-RP) are well known in the literature where as the other four (NS-Sines, NS-Gauss, CE-Gauss, CE-CRP) are novel to the best of our knowledge.

B.2 Downstream Tasks

Next, we describe the suite of downstream tasks on which POSEIDON and baselines are evaluated. The list of tasks is summarized in Table 4.

Table 4: Abbreviations/Summary for all the downstream tasks. IC and RP stand for initial conditions and Riemann problem, respectively. Datasets where (*) is checked mark datasets where solutions are learned depending on PDE parameters/sources/coefficients.

Abbreviation	PDE	(*)	Defining Feature	Visualization
NS-PwC	Navier-Stokes (31)		Piecewise constant vorticity IC	Fig. 61
NS-BB	Navier-Stokes (31)		Brownian Bridge IC	Fig. 62
NS-SL	Navier-Stokes (31)		Shear Layer IC	Fig. 63
NS-SVS	Navier-Stokes (31)		Sine Vortex sheet IC	Fig. 64
NS-Tracer-PwC	Navier-Stokes + Transport (51)		Scalar Advection	Fig. 65
FNS-KF	Forced Navier-Stokes (53)	✓	Kolmogorov Flow	Fig. 66
CE-RPUI	Euler (37)		RP with uncertain interfaces	Fig. 67
CE-RM	Euler (37)		Richtmeyer-Meshkov	Fig. 68
GCE-RT	Euler+Gravity (57)	✓	Rayleigh-Taylor	Fig. 69
Wave-Gauss	Wave Eqn. (64)	✓	Waves in Gaussian medium	Fig. 70
Wave-Layer	Wave Eqn (64)	✓	Waves in layered medium	Fig. 71
ACE	Allen-Cahn Eqn. (67)		Reaction-Diffusion	Fig. 72
SE-AF	steady state of Euler (37)	✓	Flow past airfoil	Fig. 73
Poisson-Gauss	Poisson Eqn. (68)	✓	Stationary diffusion	Fig. 74
Helmholtz	Helmholtz Eqn (69)	✓	Waves in frequency domain	Fig. 75

B.2.1 NS-PwC

This downstream task is based on the Navier-Stokes equations (31) on the space-time domain $[0,1]^2 \times [0,1]$ with periodic boundary conditions. The initial conditions are based on the vorticity, which is assumed to be constant along a uniform (square) partition of the underlying domain. To be more specific, the initial vorticity is given by,

$$\omega_0(x,y) = c_{i,j} \text{ in } [x_{i-1}, x_i] \times [y_{j-1}, y_j] \tag{41}$$

for $x_i = y_i = \frac{i}{p}$ for $i = 0, 1, 2, ..., p$, and $c_{i,j} \sim \mathcal{U}_{[-1,1]}$. The number of squares in each direction was chosen to be $p = 10$. Thus, this problem is an analogue of multiple *Riemann problems*, but on the vorticity. The underlying initial velocity field $u_x(0), u_y(0)$ is then recovered from the vorticity by using the incompressibility condition.

The underlying solution operator $\mathcal{S}(t,\cdot)$ is given by $\mathcal{S}(t, u^0_{x,y}) = u_{x,y}(t)$, with u_x, u_y solving the Navier-Stokes equations (31) with periodic boundary conditions.

We generated 20000 NS-PwC trajectories with the same train/validation/test split as NS-Sines. Note that we included 8 time steps in the training dataset, with every other time step selected, starting from step 0 up to step 14. The testing error is evaluated at the 14th time step (i.e. $t = 0.7$). A visualization of a random sample and the predictions made by POSEIDON-B, CNO and FNO by (128 training trajectories) are shown in Figure 61.

B.2.2 NS-BB

(Fractional) Brownian Bridges are widely used as an initial conditions for the Navier-Stokes equations to study statistical properties of turbulent flows in the computational physics literature, see for instance [31] and references therein.

We generate Brownian Bridges directly in Fourier space with the following method:

$$W(x) = \sum_{|\mathbf{k}|_\infty \leq N} \frac{1}{\|\mathbf{k}\|_2^{\frac{3}{2}}} \sum_{m,n,\ell \in \{0,1\}} \alpha_k^{(mn\ell)} \mathrm{sc}_m(x)\mathrm{sc}_n(x)\mathrm{sc}_\ell(x) \tag{42}$$

where

$$\mathrm{sc}_i(x) = \begin{cases} \sin(x) & \text{for } i = 0 \\ \cos(x) & \text{for } i = 1 \end{cases} \tag{43}$$

and the $\alpha_k^{(mn\ell)} \sim \mathcal{U}_{[-1,1]}$. These Brownian Bridges are propagated through the discretized Navier-Stokes system (32) from time $t = -0.5$ to $t = 0$. The resulting flow fields are then taken as initial conditions for this dataset.

The underlying solution operator $\mathcal{S}(t, \cdot)$ is given by $\mathcal{S}(t, u^0_{x,y}) = u_{x,y}(t)$, with u_x, u_y solving the Navier-Stokes equations (31) with periodic boundary conditions.

We generated 20000 NS-BB trajectories with the same train/validation/test split as NS-Sines. The same time-stepping is used as for NS-PwC. The testing error is evaluated at the 14th time step (i.e. $t = 0.7$). A visualization of a random sample and the predictions made by POSEIDON, CNO and FNO (128 training trajectories) are shown in Figure 62.

B.2.3 NS-SL

The Shear Layer (SL) is a well-known benchmark for the Navier-Stokes equations (31), stemming all the way from [5], if not earlier, see [31] for a modern (stochastic) version, whose variant we consider here.

We take as initial conditions the shear layer,

$$\begin{aligned} u_0(x, y) &= \begin{cases} \tanh\left(2\pi \frac{y-0.25}{\rho}\right) & \text{for } y + \sigma_\delta(x) \le \frac{1}{2} \\ \tanh\left(2\pi \frac{0.75-y}{\rho}\right) & \text{otherwise} \end{cases} \\ v_0(x, y) &= 0 \end{aligned} \tag{44}$$

where $\sigma_\delta : [0, 1] \to \mathbb{R}$ is a perturbation of the initial data given by

$$\sigma_\delta(x) = \xi + \delta \sum_{k=1}^{p} \alpha_k \sin(2\pi k x - \beta_k). \tag{45}$$

The parameters are chosen to be $p \sim \mathcal{U}_{\{7,8,\ldots 12\}}$ $\alpha_k \sim \mathcal{U}_{[0,1]}$, $\beta_k \sim \mathcal{U}_{[0,2\pi]}$, $\delta = 0.025$, $\rho \sim \mathcal{U}_{[0.08,0.12]}$, and $\xi \sim \mathcal{U}_{[-0.0625,0.0625]}$.

The underlying solution operator $\mathcal{S}(t, \cdot)$ is given by $\mathcal{S}(t, u^0_{x,y}) = u_{x,y}(t)$, with u_x, u_y solving the Navier-Stokes equations (31) with periodic boundary conditions.

We generated 40000 NS-SL trajectories of which the first 39640 are in the training split, the next 120 in the validation split, and the remaining 240 in the test split. The same time-stepping is used as for NS-PwC. The testing error is evaluated at the 14th time step (i.e. $t = 0.7$). A visualization of a random sample and the predictions made by POSEIDON, CNO and FNO (128 training trajectories) are shown in Figure 63.

B.2.4 NS-SVS

The *Sinusoidal Vortex Sheet* (SVS) is another classic numerical benchmark for the Navier-Stokes equations [61] and references therein. We consider a modern (stochastic) version from [31] here. The initial datum for this problem is specified in terms of the vorticity, by setting,

$$\omega_0^\rho = \psi_\rho * \omega_0 \tag{46}$$

where

$$\omega_0(x) = \delta(x - \Gamma) - \int_{\mathbb{T}^2} \mathrm{d}\Gamma \tag{47}$$

$$\phi_\rho(x) = \rho^{-2}\psi\left(\frac{\|x\|}{\rho}\right) \tag{48}$$

$$\psi(r) = \frac{80}{7\pi}\left[(r+1)_+^3 - 4(r+\frac{1}{2})_+^3 + 6r_+^3 - 4(r-\frac{1}{2})_+^3 + (r-1)_+^3\right] \tag{49}$$

$$\Gamma = \{(x,y) \in \mathbb{T}^2 \mid y = \frac{1}{2} + 0.2\sin(2\pi x) + \sum_{i=1}^{p} \alpha_i \sin(2\pi(x+\beta_i))\}. \tag{50}$$

We choose $p = 10$ and the random variables α_i and β_i are given by $\alpha_i \sim \mathcal{U}_{[0,0.003125]}$, $\beta_i \sim \mathcal{U}_{[0,1]}$. The parameter ρ is chosen to be $\rho = \frac{5}{128}$.

We generated 20000 NS-SL trajectories with the same training/validation/test split as NS-Sines. The same time-stepping is used as for NS-PwC. The testing error is evaluated at the 14th time step (i.e. $t = 0.7$). A visualization of a random sample and the predictions made by POSEIDON, CNO and FNO (128 training trajectories) are shown in Figure 64.

B.2.5 NS-Tracer-PwC

This downstream task is the first of our tasks, where the underlying physics has not been completely encountered in the pretraining dataset.

In this experiment, we focus on the important problem of transport of a passive tracer, for instance a pollutant in a river. This tracer is carried along by the Navier-Stokes flow field without feeding back into the velocity. Let $c = c(x, y, t)$ be the concentration of the passive scalar in the fluid. The equation that governs c is given by

$$\frac{\partial c}{\partial t} + \mathbf{u} \cdot \nabla c = \kappa \Delta c, \tag{51}$$

where $\mathbf{u}$ is the velocity field of the fluid, which in turn is governed by the Navier-Stokes equations (31), and κ is the diffusivity constant. We choose κ to be equal to the the artificial viscosity term used in the simulation of the flow (see B.1 for clarification).

The fluid velocity field $\mathbf{u}$ has the exact same initial data as in the NS-PwC experiment. The tracer concentration c is initialized as a sphere centered in the center of the domain

$$c_0(x, y) = \mathbb{1}_{B_{\frac{1}{4}}(\frac{1}{2},\frac{1}{2})}(x, y). \tag{52}$$

Thus, the source of stochasticity in this problem is purely the random initial condition driving the fluid flow.

The underlying solution operator $\mathcal{S}(t, \cdot)$ is now given by $\mathcal{S}(t, u_x^0, u_y^0, c_0) = [u_x(t), u_y(t), c(t)]$, with u_x, u_y solving the Navier-Stokes equations (31) with periodic boundary conditions and c solving the transport equation (51).

We generated 20000 NS-Tracer-PwC trajectories with the same train/validation/test split as NS-Sines. The same time-stepping is used as for NS-PwC. The testing error is evaluated for the 14th time step (i.e. $t = 0.7$). A visualization of a random sample and the predictions made by POSEIDON, CNO and FNO (128 training trajectories) are shown in Figure 65.

B.2.6 FNS-KF

Another downstream task which introduces a physical process that has not been encountered in the pretraining dataset, a two-dimensional version of the well-known Kolmogorov Flow [47] is modeled by Navier-Stokes equations with a forcing term, namely

$$u_t + (u \cdot \nabla)u + \nabla p - \nu \Delta u = f, \quad \operatorname{div} u = 0, \tag{53}$$

in the domain $[0, 1]^2$ with periodic boundary conditions. The forcing term f is chosen to be constant in time and is equal to

$$f(x, y) = 0.1\sin(2\pi(x + y)). \tag{54}$$

The fluid velocity field u is initialized in the exact same way as in the NS-PwC experiment. The data is simulated with the same method as the other flows governed by Navier-Stokes equations (see B.1 for clarification).

We also remark that this problem can be readily recast in the generic form (1) by considering the augmented solution vector $U = [u_x, u_y, f]$ and augmenting the PDE (1) with the trivial equation $f_t = 0$ and augmenting the initial data with (54). The underlying solution operator $\mathcal{S}(t,\cdot)$ is then given by $\mathcal{S}(t, U^0_{x,y}) = [u_x(t), u_y(t), f]$, with u_x, u_y solving the forced Navier-Stokes equations (53) with periodic boundary conditions and f being given by (54).

We generated 20000 FNS-KF trajectories with the same train/validation/test split as NS-Sines. The same time-stepping is used as for NS-PwC. The testing error is evaluated at the 14th time step (i.e. $t = 0.7$). A visualization of a random sample and the predictions made by POSEIDON, CNO and FNO (128 training trajectories) are shown in Figure 66.

B.2.7 CE-RPUI

This downstream task considers the compressible Euler equations and is a variant of the uncertain interface problem considered in [50] as well as a (hard) perturbation of CE-RP, where not just the amplitude of the jumps for each Riemann problem is randomly varied, but even the location and shape of the initial interfaces is randomly perturbed. To realize this construction, we denote the fractional part any $x \in \mathbb{R}$ as $\{x\} := x - \lfloor |x| \rfloor \operatorname{sgn} x$ and define the functions

$$\begin{aligned}\sigma_x(x,y) &= \sum_{i,j=1}^{p} \alpha_{x,i,j} \sin(2\pi(i+2p^2)x + (j+2p^2)y + \beta_{x,i,j})\\ \sigma_y(x,y) &= \sum_{i,j=1}^{p} \alpha_{y,i,j} \sin(2\pi(i+2p^2)x + (j+2p^2)y + \beta_{y,i,j}).\end{aligned}$$

where $\alpha_{k,i,j} \sim \mathcal{U}_{[-0.01,0.01]}$, and $\beta_{k,i,j} \sim \mathcal{U}_{[0,1]}$. These functions are then used to create a partitioning of the domain into subdomains

$$D_{i,j} = \{(x,y) \in \mathbb{T}^2 \mid x_{\text{min}} \le \{x + \sigma_x(x,y) + 1\} < x_{\text{max}}, y_{\text{min}} \le \{y + \sigma_y(x,y) + 1\} < y_{\text{max}}\}.$$

with $x_{\text{min}} = \frac{i}{p+1}$, $x_{\text{max}} = \frac{i+1}{p+1}$, $y_{\text{min}} = \frac{j}{p+1}$, and $y_{\text{max}} = \frac{j+1}{p+1}$. Finally, the initial conditions are given by

$$(\rho, v_x, v_y, p)|_{t=0} = (\rho_{i,j}, v^x_{i,j}, v^x_{i,j}, p_{i,j}) \text{ in } D_{i,j}$$

where $\rho_{i,j} \sim \mathcal{U}_{[1,3]}$, $v^x_{i,j} \sim \mathcal{U}_{[-10,10]}$, $v^y_{i,j} \sim \mathcal{U}_{[-10,10]}$, and $p_{i,j} \sim \mathcal{U}_{[5,7]}$.

The underlying solution operator $\mathcal{S}(t,\cdot)$ is given by $\mathcal{S}(t, \rho_0, v^0_{x,y}, p_0) = [\rho(t), v_{x,y}(t), p(t)]$ solving the compressible Euler equations (37) with periodic boundary conditions.

We generated 10000 CE-RPUI trajectories with the train/validation/test split being the same as for CE-RP. The same time-stepping is used as for NS-PwC. The testing error is evaluated at the 14th time step (i.e. $t = 0.7$). A visualization of a random sample and the predictions made by POSEIDON, CNO and FNO (128 training trajectories) are shown in Figure 67.

B.2.8 CE-RM

Another well-known benchmark for the compressible Euler equations (37) is the Richtmeyer-Meshkov problem, see [29]. A modern (stochastic) version is provided in [16]. The compressible Euler equations are considered with the initial data given by,

$$p_0(x,y) = \begin{cases} 20 & \text{if } \sqrt{x^2+y^2} < 0.1 \\ 1 & \text{otherwise.} \end{cases} \qquad \rho_0(x,y) = \begin{cases} 2 & \text{if } |x| < I(x,y,\omega) \\ 1 & \text{otherwise} \end{cases} \qquad v_0^x = w_0^y = 0 \tag{55}$$

We assign periodic boundary conditions on $D = [0,1]^2$. The interface between the two states is given as

$$I(x,y,\omega) = 0.25 + \epsilon \sum_{j=1}^{K} a_j(\omega) \sin(2\pi((x,y) + b_j(\omega))), \tag{56}$$

where $K = 10$, $\epsilon > 0$, and a_j and b_j (for $j = 1, \dots, K$) are uniform random variables on the interval $[0, 1]$. We normalize the a_j such that $\sum_j a_j = 1$. We simulate up to $T = 2$.

The underlying solution operator $\mathcal{S}(t, \cdot)$ is given by $\mathcal{S}(t, \rho_0, v^0_{x,y}, p_0) = [\rho(t), v_{x,y}(t), p(t)]$ solving the compressible Euler equations (37) with periodic boundary conditions.

We generated 1260 CE-RM trajectories with a train/validation/test split of 1030/100/130. The approximate solutions where generated with the FISH hydrodynamic code, see [16] and references therein which implements high-resolution finite volume schemes. We save 21 snapshots, evenly spaced in time. The testing error is evaluated at the 14th time step (i.e. $t = 1.4$). A visualization of a random sample and the predictions made by POSEIDON, CNO and FNO (128 training trajectories) are shown in Figure 68.

B.2.9 GCE-RT

The compressible Euler equations with gravitation (GCE) are given by,

$$\begin{aligned} u_t + \operatorname{div} F(u) = S, \; u = [\rho, \rho v, E]^{\perp}, \; F = [\rho v, \rho v \otimes v + p\mathbf{I}, (E + p)]v]^{\perp}, \\ S = [0, -\rho, 0, -\rho v_x] \frac{\partial \varphi}{\partial x} + [0, 0, -\rho, -\rho v_y] \frac{\partial \varphi}{\partial y}, \end{aligned} \tag{57}$$

with $\rho, v_{x,y}, p$ be as defined in (37) and φ being the *gravitational potential*.

For this experiment, we follow a well-known benchmark in astrophysics, namely the Rayleigh-Taylor (RT) instability on a model *neutron star*, realized as a $\gamma = 2$ polytrope in gravitational equilibrium. Our benchmark is a two-dimensional stochastic variant of the setup of [28], Section 3.2.4, with the only variation being provided by the random fields used to generate the initial conditions. The domain is $D = [-1/2, +1/2]^2$ and the pressure and gravitational potential are given by

$$p(r) = K_0 \left(\rho_0 \frac{\sin(\alpha r)}{\alpha r} \right)^2, \quad \varphi(r) = -2K_0 \rho_0 \frac{\sin(\alpha r)}{\alpha r}, \tag{58}$$

where $r = \sqrt{x^2 + y^2}$ is the radius, $K_0 = p_0/\rho_0^2$ is the polytropic constant,

$$\alpha = \sqrt{\frac{4\pi G}{2K_0}} \tag{59}$$

and $G = 1$ is the gravitational constant. The initial velocity is set to vanish. The density profile is set as

$$\rho(r) = \sqrt{\frac{K_0}{\tilde{K}(r)}} \rho_0 \frac{\sin(\alpha r)}{\alpha r}, \tag{60}$$

where

$$\tilde{K}(r) = \begin{cases} K_0, & r < r_{\mathrm{RT}} \\ \left(\frac{1-A}{1+A} \right)^2 K_0, & r \geq r_{\mathrm{RT}}. \end{cases} \tag{61}$$

Here, A is the Atwood number which parameterizes the density jump between the heavier and lighter fluid, characterizing the Rayleigh-Taylor instability. The interface between the fluids is given as

$$r_{\mathrm{RT}} = 0.25(1 + a \cos(\operatorname{atan2}(y, x) + b)), \tag{62}$$

where the amplitude a and phase b are uniform random variables on $[-1, 1]$ and $[-\pi, \pi]$, respectively. Similarly, we perturb the central density ρ_0, pressure p_0 and Atwood number as

$$\rho_0 = 1 + 0.2c, \quad p_0 = 1 + 0.2d, A = 0.05(1 + 0.2e), \tag{63}$$

where c, d, e are uniform random variables on $[-1, 1]$. We evolve the initial state up to a final time of $T = 5$ and save 11 snapshots, evenly spaced in time.

The new physical phenomena that we add in this case is *gravitational forcing* and the underlying solution operator $\mathcal{S}(t, \cdot)$ is given by $\mathcal{S}(t, \rho_0, v^0_{x,y}, p_0, \varphi) = [\rho(t), v_{x,y}(t), p(t), \varphi]$ solving the gravitational Euler equations (57) with periodic boundary conditions.

We generated 1260 GCE-RT trajectories (with the same train/validation/test split as CE-RM) with a well-balanced second-order finite volume method, as described in [28], at 256^2 resolution, then downsampled to 128^2. The testing error is evaluated at the 7th time step, and we take every snapshot up to and including the 7th as training data. A visualization of a random sample and the predictions made by POSEIDON, CNO and FNO (128 training trajectories) are shown in Figure 69.

B.2.10 Wave-Gauss

We consider the wave equation with a spatially varying propagation speed, i.e.

$$u_{tt} - (c(x))^2 \Delta u = 0, \text{ in } D \times (0, T), \tag{64}$$

in order to model the propagation of acoustic waves in a spatially varying medium.

The initial condition u_0 is given by a sum of several Gaussians whose parameters are drawn uniformly at random. First, we draw a random integer n from the set $\{2, 3, 4, 5, 6\}$. Then, for $1 \leq i \leq n$, we draw two locations, $x_{c,i}, y_{c,i} \sim \mathcal{U}_{[1/6,5/6]}$. We fix the amplitude of the ith Gaussian to 1.0 and draw the ith standard deviation $s_i \sim \mathcal{U}_{[0.039,0.156]}$ Note that we restrict any two centers of the Gaussians to be closer than 2 standard deviations from each other. If this happens, we draw a new point and discard the old one. The ith Gaussian is defined as

$$g_i(x, y) = \exp\left(-\frac{(x_{c,i} - x)^2 + (y_{c,i} - y)^2}{2s_i^2}\right), \quad x, y \in (0, 1).$$

Finally, the initial condition u_0 is defined by

$$u_0(x, y) = \sum_{i=1}^{n} g_i(x, y), \quad x, y \in (0, 1). \tag{65}$$

We use absorbing boundary conditions. The propagation speed c is spatially dependent and is generated as a sum of Gaussians in several steps. First, a random *base* speed c_0 is generated such that $c_0 \sim \mathcal{U}_{[1500,2500]}$. Then, we select 4 points in the domain, namely, $(x_1, y_1) = (0.25, 0.25)$, $(x_2, y_2) = (0.25, 0.75)$, $(x_3, y_3) = (0.75, 0.25)$ and $(x_4, y_4) = (0.75, 0.75)$. For each point i, we define a random vector (dx_i, dy_i), where $dx_i, dy_i \sim \mathcal{U}_{[-0.3125,0.3125]}$. We also draw an amplitude $v_i \sim \mathcal{U}_{[1000,2500]}$ of a Gaussian that corresponds to the i-th point, as well as its standard deviation $\sigma_i \sim \mathcal{U}_{[1/12,1/6]}$. The ith Gaussian is defined by

$$f_i(x, y) = v_i \cdot \exp\left(-\frac{(x_i + dx_i - x)^2 + (y_i + dy_i - y)^2}{2\sigma_i^2}\right), \quad x, y \in (0, 1).$$

Finally, the propagation speed is defined by

$$c(x, y) = c_0 + \sum_{i=1}^{4} f_i(x, y), \quad x, y \in (0, 1).$$

Trajectories are generated with a finite-difference method, similar to the DeVITO code [43] at 128^2 resolution. The final time of all the simulations is $T = 1$. We save 15 snapshots, evenly spaced in time.

Thus, this benchmark models the propagation of acoustic waves, generated by seismic sources, which propagate in a smoothly varying medium. The wave equation (64) can be readily recast into the generic form (1) by adding the time-derivative $v = u_t$ and the coefficient c into the solution vector $U = [u(x, t), v(x, t), c(x)]$. Thus, the differential operator in (1) can be rewritten as,

$$u_t = v, \quad v_t = c^2 \Delta u, \quad c_t = 0, \tag{66}$$

and the resulting solution operator is $\mathcal{S}(t, U_0) = [u(t), v(t), c]$.

We generated 10512 Wave-Gauss trajectories with a train/validation/test split of 10212/60/240. The same time-stepping is used as for NS-PwC. The testing error is evaluated at the 14th time step (i.e. $t = 0.7$). A visualization of a random sample and the predictions made by POSEIDON, CNO and FNO are shown in Figure 70.

B.2.11 Wave-Layer

In the Wave-Layer experiment, we also consider the wave equation with spatially dependent propagation speed (64), initial conditions given by (65). We use absorbing boundary conditions.

The propagation speed c varies spatially and is generated as a (vertically) layered medium, with each layer having a constant propagation speed drawn uniformly at random. To generate one instance of c, we first draw a random integer n from $\{3, 4, 5, 6\}$, where n represents a number of layers in c. Then, for each $2 \leq i \leq n$, we generate a $x-$dependent *frontier*, defined by

$$a_i(x) = \frac{i}{n} + c_0 + \sum_{i=1}^{10} \frac{a_i}{i} \sin(2\pi i x),$$

where, first, a_i values are drawn uniformly at random from $(0, 1)$ and then c_0 is drawn uniformly at random from $(0, 1)$ and it is rescaled by a constant that depends on i so that the adjacent frontiers are impossible to intersect. Finally, a point $(x, y) \in (0, 1)^2$ is in i-th frontier if and only if $a_i(x) \leq y \leq a_{i+1}(x)$, with $a_1 = 0$ and $a_{n+1} = 0$. Each layer i has a constant speed of propagation $c_i \sim \mathcal{U}_{[2000,5000]}$. Trajectories are generated by a finite-difference method at 128^2 resolution. The final time of all the simulations is $T = 1$. We save 21 snapshots, evenly spaced in time.

As in the Wave-Gauss benchmark, the resulting solution operator is $\mathcal{S}(t, U_0) = [u(t), v(t), c]$, with $v = u_t$ and coefficient c. The wave-layer experiment models the propagation of acoustic waves, generated by seismic sources, inside a layered subsurface medium.

We generated 10512 Wave-Layer trajectories with the same train/validation/test split as Wave-Gauss. The same time-stepping is used as for NS-PwC. The testing error is evaluated at the 14th time step (i.e. $t = 0.7$). A visualization of a random sample and the predictions made by POSEIDON, CNO and FNO are shown in Figure 71.

We remark that both the Wave-Gauss and Wave-Layer tasks are very different from the pretraining dataset as the wave equation is a linear second-order (in time and space) equation that is different from the compressible Euler and incompressible Navier-Stokes equations that form the pretraining dataset.

B.2.12 ACE

The Allen-Cahn equation for modeling phase transitions in material science is given by

$$u_t = \Delta u - \epsilon^2 u(u^2 - 1), \tag{67}$$

with a reaction rate of $\epsilon = 220$. We consider this equation with periodic boundary conditions and initial conditions given by

$$u_0(x, y) = \frac{1}{K^2} \sum_{i,j=1}^{K} a_{ij} \cdot (i^2 + j^2)^{-r} \sin(\pi i x) \sin(\pi j y), \quad \forall x, y \in (0, 1),$$

where K is a random integer drawn uniformly at random from $[16, 32]$, $r \sim \mathcal{U}_{[0.7,1.0]}$ and $a_{ij} \sim \mathcal{U}_{[-1,1]}$.

Trajectories are generated by a finite-difference method at 128^2 resolution. The final time of all the simulations is $T = 0.0002$. We save 20 snapshots, evenly spaced in time.

The corresponding solution operator is $\mathcal{S}(t, u_0) = u(t)$ and maps the initial concentration to the concentration at time t.

We generated 15000 ACE trajectories with a train/validation/test split of 14700/60/240. The same time-stepping is used as for NS-PwC. The testing error is evaluated at the 14th time step. A visualization of a random sample and the predictions made by POSEIDON, CNO and FNO (128 training trajectories) are shown in Figure 72.

Again, it is essential to emphasize that the Allen-Cahn equation is a nonlinear parabolic reaction-diffusion equation that is very different from the PDEs used in constructing the pretraining dataset.

B.2.13 SE-AF

This dataset contains the samples that describe the classical computational physics benchmark of flow past airfoils, modeled by the compressible Euler equations (37). The samples are *not* time-dependent, as we are interested in the *steady-state* solution.

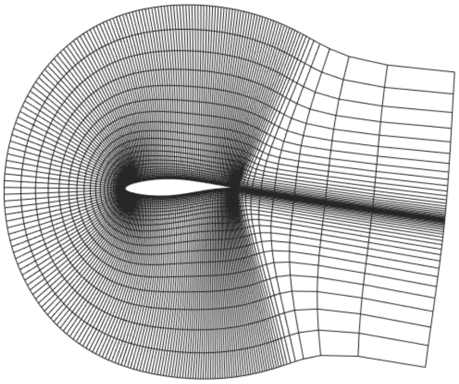


Figure 3: Elliptic mesh for the airfoil problem

We follow standard practice in aerodynamic shape optimization and consider a reference airfoil shape with upper and lower surface of the airfoil are located at $(x, y^{\mathrm{U}}_{\mathrm{ref}}(x/c))$ and $(x, y^{\mathrm{L}}_{\mathrm{ref}}(x/c))$ where c is the chord length and $y^{\mathrm{U}}_{\mathrm{ref}}$ and $y^{\mathrm{L}}_{\mathrm{ref}}$ corresponding to the well-known RAE2822 airfoil [46]. The reference shape is then perturbed by *Hicks-Henne Bump functions* [48] :

$$y^{\mathrm{L}}(\xi) = y^{\mathrm{L}}_{\mathrm{ref}}(\xi) + \sum_{i=1}^{15} a^{\mathrm{L}}_i B_i(\xi), \quad y^{\mathrm{U}}(\xi) = y^{\mathrm{U}}_{\mathrm{ref}}(\xi) + \sum_{i=1}^{15} a^{\mathrm{U}}_i B_i(\xi),$$

$$B_i(\xi) = \sin^3(\pi \xi^{q_i}), \quad q_i = \frac{\ln 2}{\ln 14 - \ln i}, \quad \xi = \frac{x}{c},$$

$$a^{\mathrm{L}}_i = 2(\psi_i - 0.5)(i+1) \times 10^{-3}, \quad a^{\mathrm{U}}_i = 2(\psi_{i+10} - 0.5)(11-i) \times 10^{-3}, \quad i = 1, ..., 15$$

with $\psi \in [0,1]^d$. We can now formally define the airfoil shape as $\mathcal{S} = \{(x,y) \in D : x \in [0,c], y^L \leq y \leq y^U\}$ and accordingly the shape function $f = \chi_{[\mathcal{S}]}(x,y)$, with χ being the *characteristic function*.

The underlying operator of interest maps the shape function f into the density of the flow ρ at steady state of the compressible Euler equations.

The equations are solved with the solver NUWTUN, see [45] and references therein, on 243×43 elliptic mesh (see Figure 3) given the following free-stream boundary conditions,

$$T^{\infty} = 1, \quad M^{\infty} = 0.729, \quad p^{\infty} = 1, \quad \alpha = 2.31^{\circ}.$$

The data is ultimately interpolated onto a Cartesian grid of dimensions 128×128 on the underlying domain $D = [-0.75, 1.75]^2$, and unit values are assigned to the density $\rho(x,y)$ for all (x,y) in the set $\mathcal{S}$. The shapes of the training data samples correspond to 30 bump functions, with coefficients ψ sampled uniformly from $[0,1]^{30}$. During the training and evaluation processes, the difference between the learned solution and the ground truth is exclusively calculated for the points (x,y) that do not belong to the airfoil shape $\mathcal{S}$.

We generated 10869 SE-AF solutions with a train/validation/test split of 10509/120/240. A visualization of a random sample and the predictions made by POSEIDON, CNO and FNO (128 training samples) are shown in Figure 73.

We note here that this SE-AF benchmark differs from what has been seen during pretraining in many aspects, namely i) the problem is time-independent, in contrast to the time-dependent PDEs for pretraining, ii) the solution operator is very different as it maps a (shape) coefficient into the steady state solution, and iii) the geometry of the underlying domain is non-Cartesian and the boundary conditions are very different from what was encountered during pretraining.

B.2.14 Poisson-Gauss

We consider the Poisson equation,

$$-\Delta u = f, \text{ in } (0,1)^2, \tag{68}$$

with homogeneous Dirichlet boundary conditions. The solution operator maps the source term f to the solution u. The source term f consists of a superposition of a random number of Gaussians

$$f(x, y) = \sum_{i=1}^{N} \exp\left(-\frac{(x-\mu_{x,i})^2 + (y-\mu_{y,i})^2}{2\sigma_i^2}\right)$$

with N being an integer drawn from a geometric distribution Geom(0.4), $\mu_{x,i}, \mu_{y,i} \sim \mathcal{U}_{[0,1]}$ and $\sigma_i \sim \mathcal{U}_{[0.025,0.1]}$. Thus, this experiment models the diffusion of an input (source) which is a superposition of Gaussians.

We generated 20000 Poisson-Gauss solutions (with a train/validation/test split of 19640/120/240) with a finite element method based on FENICS [40]. A visualization of a random sample and the predictions made by POSEIDON, CNO and FNO (128 training samples) are shown in Figure 74.

We note here that both the Poisson-Gauss and Helmholtz benchmarks differ from what has been seen during pretraining in many aspects, namely i) the problems are time-independent, in contrast to the time-dependent PDEs for pretraining, ii) the solution operator is very different as it maps coefficients into the steady-state solution, and iii) the boundary conditions are very different from the periodic boundary conditions, seen during pretraining.

B.2.15 Helmholtz

The Helmholtz equation models wave propagation in the frequency domain. We consider a variant of this equation given by

$$-\Delta u - \omega^2 a(x, y)u = 0, \quad x, y \in D, \tag{69}$$

and Dirichlet boundary conditions

$$u(x, y) = b, \quad x, y \in \partial D,$$

where $\omega = 5\pi/2$ is the frequency, $D = (0, 1)^2$ is the domain, a is the spatial dependent function that defines properties of the medium of the wave propagation and b is the fixed value of the solution u at the boundary ∂D.

The boundary value b follows uniform ditribution, namely $b \sim \mathcal{U}_{[0.25,0.5]}$. The function a is defined as a sum of random number of Gaussians and is generated in several steps. First, we draw an integer n from $[2, 7]$ uniformly at random. This number represents the number of Gaussians that will be randomly generated. For $1 \leq i \leq n$, it holds that $A_i \sim \mathcal{U}_{[0.5,10.0]}$ and $\sigma_i \sim \mathcal{U}_{[0.05,0.1]}$. Additionally, two numbers x_i, y_i that represent x and y coordinates of the Gaussians are generated such that $x_i, y_i \sim \mathcal{U}_{[0.2,0.8]}$. The unnormalized function $\bar{a}$ is obtained by

$$\bar{a}(x, y) = -\sum_{i=1}^{n} A_i \exp\left(-\frac{(x_i - x)^2 + (y_i - y)^2}{2\sigma_i^2}\right), \quad x, y \in (0, 1).$$

Function a is obtained by normalizing $\bar{a}$, i.e.

$$a(x, y) = \frac{\bar{a} - \min(\bar{a})}{\max(\bar{a}) - \min(\bar{a})}.$$

The solution operator maps the tuple (a, b) to the solution u. This problem is a *steady-state* problem. Trajectories are generated by a finite-difference method at 128^2 resolution, similar to DeVito [43].

We generated 19675 Helmholtz solutions with a train/validation/test split of 19035/128/512. A visualization of a random sample and the predictions made by POSEIDON, CNO and FNO (128 training trajectories) are shown in Figure 75.

C Models and Baselines

We compare multiple versions of Poseidon with foundation model baselines, namely MPP [49] and a CNO [60] foundation model that is trained in a similar manner as Poseidon. In addition, we compare against state-of-the-art neural operators trained from scratch, namely CNO and FNO [33], as well as scOT trained from scratch. All these models and their training strategies are described in the following. Their approximate model sizes can be read off from Table 5.

Table 5: Approximate model sizes of all the models considered in this paper.

Model	Number of parameters
Poseidon-L	629M
Poseidon-B	158M
Poseidon-T	21M
CNO-FM	109M
MPP-B	116M
CNO	39M
scOT	40M
FNO	37M

We train all models on a realization of Equation 6 in the Main Text; in particular, we set $p = 1$. For each gradient step, we draw from the set of N available trajectory snapshots $\{\mathbf{u}^l_{t_k} | \mathbf{u}^l_{t_k} \in \mathbb{R}^{c \times J \times J}\}_{l=1}^N$ where c is the number of input/output functions, J is the size of the computational grid, and t_k is the (lead) time from the initial condition to the k-th snapshot in the trajectory, i.e. we get a batch of size B of pairs $\{(\mathbf{u}^m_{t_i}, \mathbf{u}^m_{t_j})_l\}_{l=1}^B$ where $i \leq j$. The loss is then computed as

$$\mathcal{L}(\{(\mathbf{u}^m_{t_i}, \mathbf{u}^m_{t_j})_l\}_{l=1}^B) = \frac{1}{c} \sum_{s=1}^{c} \frac{\sum_{l=1}^B \sum_{u,v=1}^J \left| (\mathbf{u}^m_{t_j})^l_{s,u,v} - \mathcal{S}_\theta((t_j)^l - (t_i)^l, (\mathbf{u}^m_{t_i})^l_{s,u,v}) \right|}{\sum_{l=1}^B \sum_{u,v=1}^J \left| (\mathbf{u}^m_{t_j})^l_{s,u,v} \right| + \epsilon} \tag{70}$$

where $\epsilon = 10^{-10}$ for numerical stability, $\mathcal{S}_\theta$ is the model, $(\cdot)^l$ is the l-th sample from the batch and $(\cdot)_{s,u,v}$ denotes the value at indices (s, u, v).

During training, we create a checkpoint after every epoch, but only keep the checkpoint corresponding to the lowest validation loss (evaluated at the end of the epoch) which is then also used for testing.

C.1 Poseidon Models

In the following, we give thorough details for all Poseidon models that we pretrained, as well as details on finetuning these pretrained models. All models are pretrained on the datasets introduced in Section B.1, i.e. they expect four dimensional inputs and outputs, density ρ, velocities u and v, and pressure p. During pretraining, we set $\rho = 1$ and mask p for all pretraining datasets corresponding to incompressible flow. Further, we use the full set of 77840 pretraining trajectories, unless otherwise specified.

To finetune the pretrained model on tasks whose input/output functions are not in the set of pretraining input/outputs (ρ, u, v, p) or where there are additional inputs/outputs – this corresponds to the tasks NS-Tracer-PwC, FNS-KF, SE-AF, GCE-RT, Wave-Layer, Wave-Gauss, ACE, Poisson-Gauss, and Helmholtz – we transfer all parameters from the pretrained model, except

- the embedding weight $\mathbf{W}_\mathcal{E}$,
- the patch recovery weight $\mathbf{W}_\mathcal{R}$ and bias $\mathbf{b}_\mathcal{R}$, and
- the mixup convolutional kernel.

We refer to Section A.2 for notation. This means that solely embedding/recovery is trained from scratch and just the parameters whose dimensions would not match, i.e. *a minimal set of parameters* is trained from random initialization. All other parameters are kept trainable and *no parameter is frozen*.

In general, we do not apply any weight decay on (time-conditioned) layer norm parameters. We finetune all parameters using the same optimizer with different learning rates, i.e. we build two or three parameter groups, depending on the finetuning task. In case the embedding and recovery do not have to be replaced, we finetune all parameters, except parameters of the (time-conditioned) layer norm, with learning rate $\widehat{\eta}$, and parameters of the layer norm with learning rate $\widetilde{\eta}_{\mathcal{N}}$. If the embedding/recovery is to be replaced and trained from scratch, we finetune all embedding/recovery parameters (including the embedding bias $\mathbf{b}_{\mathcal{E}}$) with learning rate $\widetilde{\eta}$, layer norm parameters with learning rate $\widetilde{\eta}_{\mathcal{N}}$, and all other parameters with learning rate $\widehat{\eta}$.

C.1.1 POSEIDON-T

POSEIDON-T is the smallest pretrained model, an instantiated scOT with the following hyperparameters:

- **Embedding/latent dimension** C: 48
- **Number of SwinV2 transformer blocks at each level** $(\forall i)$ t_i: 4

This results in a model with 21M parameters (for POSEIDON models, we exclude embedding and recovery parameters in this count).

Pretraining The model is pretrained on 8 NVIDIA RTX 4090 GPUs using the following (data-parallel) training protocol:

- **Optimizer:** AdamW [41]
- **Scheduler:** Cosine Decay with linear warmup of 2 epochs
- **Maximum learning rate:** 10^{-3}
- **Weight decay:** 0.1
- **Effective batch size:** 640, resulting in a per-device batch size of 80
- **Number of epochs:** 40
- **Early stopping:** No
- **Gradient clipping (maximal norm):** 5

Finetuning The pretrained model is finetuned on every task on a single GPU following this finetuning protocol ($\widetilde{\eta}$ is only applicable to certain downstream tasks):

- **Optimizer:** AdamW [41]
- **Scheduler:** Cosine Decay
- **Initial learning rate** $\widehat{\eta}$: $5 \cdot 10^{-5}$
- **Initial learning rate** $\widetilde{\eta}$: $5 \cdot 10^{-4}$
- **Initial learning rate** $\widetilde{\eta}_{\mathcal{N}}$: $5 \cdot 10^{-4}$
- **Weight decay:** 10^{-6}
- **Batch size:** 40
- **Number of epochs:** 200
- **Early stopping:** No
- **Gradient clipping (maximal norm):** 5

C.1.2 POSEIDON-B

POSEIDON-B is the base model, an instantiated scOT with the following hyperparameters:

- **Embedding/latent dimension** C: 96
- **Number of SwinV2 transformer blocks at each level** $(\forall i)$ t_i: 8

This results in a model with 158M parameters.

Pretraining The model is pretrained on 8 NVIDIA RTX 4090 GPUs using the following (data-parallel) training protocol:

- **Optimizer:** AdamW [41]
- **Scheduler:** Cosine Decay with linear warmup of 2 epochs
- **Maximum learning rate:** $5 \cdot 10^{-4}$
- **Weight decay:** 0.1
- **Effective batch size:** 320, resulting in a per-device batch size of 40
- **Number of epochs:** 39 (40 were initially planned)
- **Early stopping:** No
- **Gradient clipping (maximal norm):** 5

Finetuning The pretrained model is finetuned on every task on a single GPU following this finetuning protocol ($\widetilde{\eta}$ is only applicable to certain downstream tasks):

- **Optimizer:** AdamW [41]
- **Scheduler:** Cosine Decay
- **Initial learning rate** $\widehat{\eta}$: $5 \cdot 10^{-5}$
- **Initial learning rate** $\widetilde{\eta}$: $5 \cdot 10^{-4}$
- **Initial learning rate** $\widetilde{\eta}_{\mathcal{N}}$: $5 \cdot 10^{-4}$
- **Weight decay:** 10^{-6}
- **Batch size:** 40
- **Number of epochs:** 200
- **Early stopping:** No
- **Gradient clipping (maximal norm):** 5

C.1.3 POSEIDON-L

POSEIDON-L is the largest model we trained, an instantiated scOT with the following hyperparameters:

- **Embedding/latent dimension** C: 192
- **Number of SwinV2 transformer blocks at each level** $(\forall i)$ t_i: 8

This results in a model with 629M parameters.

Pretraining The model is pretrained on 8 NVIDIA RTX 4090 GPUs using the following (data-parallel) training protocol:

- **Optimizer:** AdamW [41]
- **Scheduler:** Cosine Decay with linear warmup of 1 epoch
- **Maximum learning rate:** $2 \cdot 10^{-4}$
- **Weight decay:** 0.1
- **Effective batch size:** 128, resulting in a per-device batch size of 16
- **Number of epochs:** 20
- **Early stopping:** No
- **Gradient clipping (maximal norm):** 5

Finetuning The pretrained model is finetuned on every task on a single GPU following this finetuning protocol ($\widetilde{\eta}$ is only applicable to certain downstream tasks):

- **Optimizer:** AdamW [41]
- **Scheduler:** Cosine Decay
- **Initial learning rate** $\widehat{\eta}$: $5 \cdot 10^{-5}$
- **Initial learning rate** $\widetilde{\eta}$: $5 \cdot 10^{-4}$
- **Initial learning rate** $\widetilde{\eta}_{\mathcal{N}}$: $5 \cdot 10^{-4}$
- **Weight decay:** 10^{-6}
- **Batch size:** 16
- **Number of epochs:** 200
- **Early stopping:** No
- **Gradient clipping (maximal norm):** 5

C.1.4 Models for Dataset Ablations (see Section D.3)

For models used in the pretraining dataset ablations, we utilize the same pretraining and finetuning strategies as for POSEIDON-B. For the model trained on half of the pretraining dataset, we only train on the first half of each subset (NS-Sines, NS-Gaussians, CE-RP, CE-CRP, CE-KH, CE-Gauss); the same logic applies to the model trained on an eighth of the pretraining dataset. The model trained on a less diverse pretraining dataset is not trained on NS-Sines, CE-CRP, and CE-Gauss, such that the pretraining dataset size is directly comparable to the model trained on half of the pretraining dataset.

C.2 scOT

We additionally train a scOT from scratch on every downstream task, to compare its performance to POSEIDON and other baselines. Its hyperparameters are as follows:

- **Embedding/latent dimension** C: 48
- **Number of SwinV2 transformer blocks at each level** $(\forall i)$ t_i: 8

This results in a model with 40M parameters. It is trained on one or multiple GPUs (depending on the dataset size) with the following parameters:

- **Optimizer:** AdamW [41]
- **Scheduler:** Cosine Decay with linear warmup of 20 epochs
- **Maximum learning rate** $\widehat{\eta}$: $5 \cdot 10^{-4}$
- **Weight decay:** 10^{-6}
- **Batch size:** 40 (on a single GPU, else the effective batch size is larger)
- **Number of epochs:** 400
- **Early stopping:** If the validation loss does not improve for 40 epochs
- **Gradient clipping (maximal norm):** 5

C.3 CNO

A *Convolutional Neural Operator* (CNO) is a model that (approximately) maps bandlimited functions to bandlimited functions [60]. Let $\mathcal{B}_w$ be the space of bandlimited functions with the bandlimit w. A CNO is compositional mapping between function spaces $\mathcal{G} : \mathcal{B}_w(D) \to \mathcal{B}_w(D)$ and is defined as

$$\mathcal{G} : u \mapsto P(u) = v_0 \mapsto v_1 \mapsto \dots v_L \mapsto Q(v_L) = \bar{u}, \tag{71}$$

where

$$v_{l+1} = \mathcal{P}_l \circ \Sigma_l \circ \mathcal{K}_l(v_l), \quad 1 \leq \ell \leq L-1, \tag{72}$$

where L is the number of CNO blocks and $D = (0,1)^2$ is the domain.

First, the input function $u \in \mathcal{B}_w(D)$ is lifted to the latent space of bandlimited functions through a *lifting layer*:

$$P : \left\{u \in \mathcal{B}_w(D, \mathbb{R}^{d_{\mathcal{X}}})\right\} \to \left\{v_0 \in \mathcal{B}_w(D, \mathbb{R}^{d_0})\right\}.$$

Here, $d_0 \geq d_{\mathcal{X}}$ is the number of channels in the lifted, latent space. The lifting operation is performed by a convolution operator and activation operator which will be defined below.

Then, the lifted function is processed through the composition of a series of mappings between functions (layers), with each layer consisting of three elementary mappings, i.e., $\mathcal{P}_l$ is either the *upsampling* or *downsampling* operator, $\mathcal{K}_l$ is the convolution operator and Σ_l is the activation operator.

Finally, the last output function in the iterative procedure v_L is projected to the output space with a *projection operator* Q, defined as

$$Q : \left\{v_L \in \mathcal{B}_w(D, \mathbb{R}^{d_L})\right\} \to \left\{\overline{u} \in \mathcal{B}_w(D, \mathbb{R}^{d_{\mathcal{Y}}})\right\}.$$

The projection operation is also performed by a convolution operator and activation operator.

Upsampling and Downsampling Operators. For some $\overline{w} > w$, we can *upsample* a function $f \in \mathcal{B}_w$ to the *higher band* $\mathcal{B}_{\overline{w}}$ by simply setting,

$$\mathcal{U}_{w,\overline{w}} : \mathcal{B}_w(D) \to \mathcal{B}_{\overline{w}}(D), \quad \mathcal{U}_{w,\overline{w}} f(x) = f(x), \quad \forall x \in D.$$

On the other hand, for some $\underline{w} < w$, we can *downsample* a function $f \in \mathcal{B}_w$ to the *lower band* $\mathcal{B}_{\underline{w}}$ by setting $\mathcal{D}_{w,\underline{w}} : \mathcal{B}_w(D) \to \mathcal{B}_{\underline{w}}(D)$, defined by

$$\mathcal{D}_{w,\underline{w}} f(x) = \left(\frac{\underline{w}}{w}\right)^2 (h_{\underline{w}} \star f)(x) = \left(\frac{\underline{w}}{w}\right)^2 \int_D h_{\underline{w}}(x-y) f(y) dy, \quad \forall x \in D,$$

where $\star$ is the convolution operation on functions defined above and $h_{\underline{w}}$ is the so-called *interpolation sinc filter*:

$$h_w(x_0, x_1) = \text{sinc}(2wx_0) \cdot \text{sinc}(2wx_1), \quad (x_0, x_1) \in \mathbb{R}^2. \tag{73}$$

Activation Operator. First, the input function $f \in \mathcal{B}_w$ is upsampled to a higher bandlimit $\overline{w} > w$, then the activation function is applied and finally the result is downsampled back to the original bandlimit w. Implicitly assuming that $\overline{w}$ is large enough such that $\sigma\left(\mathcal{B}_w\right) \subset \mathcal{B}_{\overline{w}}$, we define the activation operator in (71) as,

$$\Sigma_{w,\overline{w}} : \mathcal{B}_w(D) \to \mathcal{B}_w(D), \quad \Sigma_{w,\overline{w}} f(x) = \mathcal{D}_{\overline{w},w}(\sigma \circ \mathcal{U}_{w,\tilde{w}} f)(x), \quad \forall x \in D. \tag{74}$$

The above ingredients are assembled together in the form of an Operator U-Net architecture that has bandlimited functions as inputs and outputs. In addition to the blocks that have been defined above, one also needs additional ingredients, namely incorporate *skip connections* through *ResNet* blocks of the form, $\mathcal{R}_{w,\overline{w}} : \mathcal{B}_w(D, \mathbb{R}^d) \to \mathcal{B}_w(D, \mathbb{R}^d)$ such that

$$\mathcal{R}_{w,\overline{w}}(v) = v + \mathcal{K}_w \circ \Sigma_{w,\overline{w}} \circ \mathcal{K}_w(v), \quad \forall v \in \mathcal{B}_w(D, \mathbb{R}^d). \tag{75}$$

Additionally, the so-called *Invariant blocks* of the form, $\mathcal{I}_{w,\overline{w}} : \mathcal{B}_w(D, \mathbb{R}^d) \to \mathcal{B}_w(D, \mathbb{R}^d)$ is defined such that

$$\mathcal{I}_{w,\overline{w}}(v) = \Sigma_{w,\overline{w}} \circ \mathcal{K}_w(v), \quad \forall v \in \mathcal{B}_w(D, \mathbb{R}^d). \tag{76}$$

Finally, all these ingredients are assembled together in a modified Operator U-Net architecture which is graphically depicted in Figure 4. Note that instead of a *lead-time conditioned* layer normalization 4, we incorporate a *lead-time conditioned instance normalization* into CNO. A lead-time conditioned instance normalization is applied to an input $\mathbf{v}$ by

$$IN_{\alpha(t),\beta(t)}(\mathbf{v})(x) = \alpha(t) \odot IN(\mathbf{v})(x) + \beta(t) \tag{77}$$

where $IN(\mathbf{v})$ is a regular instance normalization. In the case of CNO, we use (small) MLPs to parametrize $\alpha(t)$ and $\beta(t)$. This choice of conditional layer is similar to the FILM layer introduced in [55], applied on top of the instance normalization. Additionally, we observed that *including time t* as an additional, constant input channel of the CNO slightly enhances its performance.

The specifications of the CNO model that we used and trained from scratch in all the experiments, as well as the training details are summarized in the following list:

- **Lifting dimension:** 54
- **Number of up/downsampling layers:** 4
- **Number of residual blocks in the bottleneck:** 6
- **Number of residual blocks in the middle layers:** 6
- **Trainable parameters:** 39.1M
- **Optimizer:** AdamW [41]
- **Scheduler:** Linear with decreasing factor of 0.9 every 10 epochs
- **Initial learning rate:** $5 \cdot 10^{-4}$
- **Weight decay:** 10^{-6}
- **Number of epochs:** 400
- **Batch size:** 32
- **Early stopping:** If the validation loss does not improve for 40 epochs

Source code for CNO is available at https://github.com/camlab-ethz/ConvolutionalNeuralOperator.

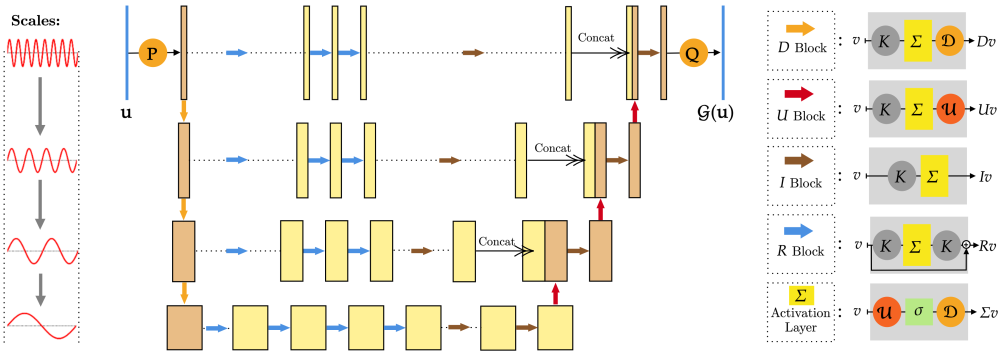


Figure 4: Schematic representation of CNO (71) as a modified U-Net with a sequence of layers mapping between bandlimited functions.

C.4 FNO

A *Fourier neural operator* (FNO) $\mathcal{G}$ [33] is a composition

$$\mathcal{G} : \mathcal{X} \to \mathcal{Y} : \quad \mathcal{G} = Q \circ \mathcal{L}_T \circ \dots \circ \mathcal{L}_1 \circ R. \tag{78}$$

It has a "lifting operator" $u(x) \mapsto R(u(x), x)$, where R is represented by a linear function $R : \mathbb{R}^{d_u} \to \mathbb{R}^{d_v}$ where d_u is the number of components of the input function and d_v is the "lifting dimension". The operator Q is a non-linear projection, instantiated by a shallow neural network with a single hidden layer and leaky ReLU activation function, such that $v^{L+1}(x) \mapsto \mathcal{G}(u)(x) = Q\left(v^{L+1}(x)\right)$.

Each *hidden layer* $\mathcal{L}_\ell : v^\ell(x) \mapsto v^{\ell+1}(x)$ is of the form

$$v^{\ell+1}(x) = (\sigma \circ IN)\left(W_\ell \cdot v^\ell(x) + \left(K_\ell v^\ell\right)(x)\right),$$

with $W_\ell \in \mathbb{R}^{d_v \times d_v}$ a trainable weight matrix (residual connection), σ an activation function, corresponding to leaky ReLU, IN standard instance normalization or time-conditioned instance normalization (see Equation 77) and the *non-local Fourier layer*,

$$K_\ell v^\ell = \mathcal{F}_N^{-1}\left(P_\ell(k) \cdot \mathcal{F}_N v^\ell(k)\right),$$

where $\mathcal{F}_N v^\ell(k)$ denotes the (truncated)-Fourier coefficients of the discrete Fourier transform (DFT) of $v^\ell(x)$, computed based on the given J grid values in each direction. Here, $P_\ell(k) \in \mathbb{C}^{d_v \times d_v}$ is a complex Fourier multiplication matrix indexed by $k \in \mathbb{Z}^d$, and $\mathcal{F}_N^{-1}$ denotes the inverse DFT. As with CNO (Section C.3), we include time as an additional channel – in addition to the time-conditioned instance normalization layers – for all time-dependent problems.

We used the following hyperparameters and training details to train the FNO models:

- **Lifting dimension:** 96
- **Number of Fourier layers:** 5
- **Number of Fourier modes:** 20
- **Trainable parameters:** 37.0M
- **Optimizer:** AdamW [41]
- **Scheduler:** Cosine Decay
- **Initial learning rate:** $5 \cdot 10^{-4}$
- **Weight decay:** 10^{-6}
- **Number of epochs:** 400
- **Batch size:** 40
- **Early stopping:** If the validation loss does not improve for 40 epochs

C.5 CNO-FM

In addition to the POSEIDON models, we also pretrain a CNO foundation model baseline. We use the same pretraining datasets as the POSEIDON models (see B.1), i.e. NS-Sines, NS-Gauss, CE-RP, CE-KH, CE-CRP and CE-Gauss datasets. The inputs and the outputs of the model have 4 channels, i.e. ρ, u, v and p. For the NS-Sines and NS-Gauss datasets, we mask out the pressure predictions during training, while predicting a constant value $\rho = 1$ for density.

The specifications of the CNO-FM model that we pretrained, as well as the training details are summarized in the following list:

- **Lifting dimension:** 82
- **Number of up/downsampling layers:** 4
- **Number of residual blocks in the bottleneck:** 8
- **Number of residual blocks in the middle layers:** 8
- **Trainable parameters:** 109M
- **Optimizer:** AdamW
- **Scheduler:** Linear with decreasing factor of 0.9 every epoch
- **Initial learning rate:** $5 \cdot 10^{-4}$
- **Weight decay:** 10^{-6}
- **Effective batch size:** 256, resulting in a per-device batch size of 32
- **Number of epochs:** 40
- **Early stopping:** No

To finetune the CNO-FM, we differentiate between two scenarios: one where the input and output share the same context as the pretrained models (comprising the variables ρ, u, v, and p, either masked or unmasked), and another where the downstream task is out-of-context (i.e. when the input and target variables differ from those used during pretraining).

To explain the finetuning technique, let us denote the pretrained CNO model by $\mathcal{G}_{FM}$ and decompose it to

$$\mathcal{G}_{FM} = Q \circ \mathcal{G}_{FM,b} \circ P,$$

where P is the lifting layer, Q is the projection layer and $\mathcal{G}_{FM,b}$ is the base part of the CNO-FM.

When the context of variables is retained in the downstream task, we introduce an additional linear layer $\mathcal{L}$ that is applied prior to the lifting layer P. All other parameters from $\mathcal{G}_{FM}$ are transferred over to the downstream task model. Hence, the model that is finetuned is

$$\mathcal{G}_{FT} = Q \circ \mathcal{G}_{FM,b} \circ P \circ \mathcal{L}. \tag{79}$$

A schematic representation of the CNO-FM finetuning procedure is shown in Figure 5. When the downstream task is out-of-context, in addition to the linear layer $\mathcal{L}$ that is applied before P, the projection layer is replaced by a new, randomly initialized projection layer $Q^\star$. Other parameters are transferred over to the downstream task model. The model that is finetuned is

$$\mathcal{G}_{FT} = Q^\star \circ \mathcal{G}_{FM,b} \circ P \circ \mathcal{L}. \tag{80}$$

We set the number of epochs for the downstream tasks to 200. Since the loss converges significantly faster than when training from scratch, even $50 - 100$ epochs were sufficient to effectively finetune the CNO-FM. Parameters of $\mathcal{G}_{FT}$ are divided into three distinct groups

- **Group 1**: Projection Q (or $Q^\star$), Lifting P and Linear layer $\mathcal{L}$
- **Group 2**: All the conditional instance normalization layers $IN_{\alpha(t),\beta(t)}$
- **Group 3**: Other parameters in $\mathcal{G}_{FM,b}$

We experimented with learning rates for each group of parameters, as well as schedulers. An efficient way to finetune CNO-FM for in-context downstream tasks was to set the initial learning rates of the parameter groups to $lr_1 = 2.5 \cdot 10^{-4}$, $lr_2 = 5 \cdot 10^{-4}$ and $lr_3 = 10^{-4}$. For out-of-context tasks, the learning rates that we used are $lr_1 = 7.5 \cdot 10^{-4}$, $lr_2 = 5 \cdot 10^{-4}$ and $lr_3 = 10^{-4}$. In both cases, the learning rate scheduler is linear with with decreasing factor of 0.9 every 5 epochs.

The CNO codes are available at https://github.com/camlab-ethz/ConvolutionalNeuralOperator.

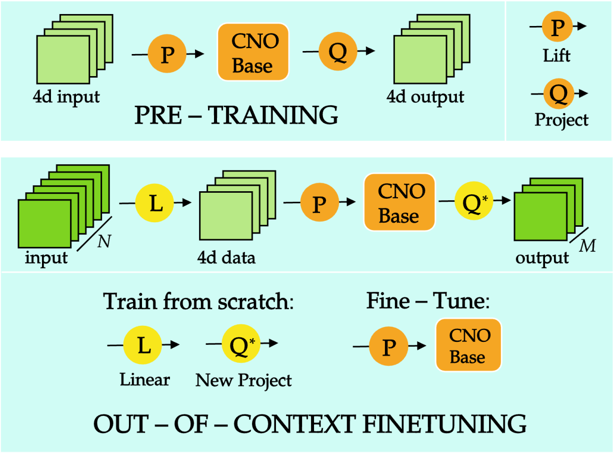


Figure 5: Schematic representation of the finetuning procedure of CNO-FM.

C.6 MPP

Multiple physics pretraining (MPP) is a pretraining approach for autoregressive physical surrogate modeling [49]. MPP uses a *scalable axial attention* transformer backbone to reduce the quadratic complexity of the usual attention mechanism. Multiple input fields of MPP are projected onto a single, shared embedding space. MPP also uses spatial and temporal attention blocks to capture spatial and temporal dependencies in the data. To train or finetune MPP models, one uses the normalized MSE loss. We will finetune the **MPP-AVIT-B** foundation model for all our downstream tasks. The **MPP-AVIT-B** model has 116M trainable parameters.

MPP models are autoregressive models with fixed context size of T^S. They predict the solution at a time step N of a PDE of interest given the previous T^S time steps. Thus, they rely on the *history of the solution*, encompassing multiple time steps, to forecast future time steps accurately. This differs from the the task that we are interested in (i.e. **OLT** defined in the Main Text), which aims to generate the entire solution trajectory given only the initial datum and boundary conditions.

Therefore, we need to adjust the MPP finetuning strategy. We adapt the *all2all* strategy. Let $U = (u_0, u_1, \dots, u_T)$ be a solution trajectory of length $T + 1$. Let (i, j) be two integers such that $j > i$. We rely on the fact that MPP predicts one snapshot at a time and finetune MPP to predict u_j based on the history $u_{j-1}, u_{j-2}, \dots u_i$. Since there are not always T^S past time steps in the training samples, we fill the remaining time steps with copies of u_i (see Figure 6). We generate $T(T+1)/2$ samples out of the trajectory U. For steady-state operators of the form $(f_1, f_2, \dots, f_L) \to u$, the L channels are copied T^S times, and MPP is finetuned using these samples (see Figure 6). The inference strategy for the time-dependent problems is straightforward. Given the initial snapshot u_0, one autoregressively applies the finetuned model T times to predict u_T (see Figure 6). During the finetuning of MPP, we do not predict dummy variables like the speed function in the Wave equation or the forcing term in Kolmogorov flow, as the model had difficulties in predicting them, so the errors accumulated fast. This contrasts with other models that predict the dummy variables alongside the solution. Final testing errors for all the models are not calculated for these dummy variables.

For each downstream task, we finetuned **MPP-AVIT-B** model for 100 epochs. We did not use more than 100 epochs as the training usually converged after 10 to 50 epochs. We used the Adam [25] optimizer with a cosine annealing scheduler and linear warmup.

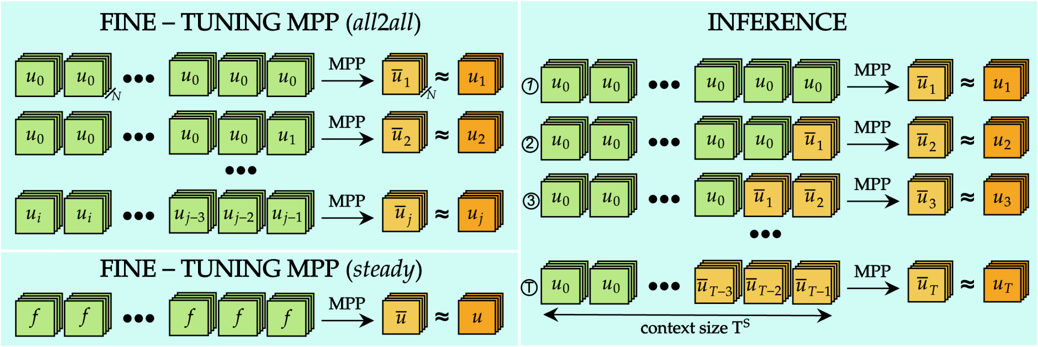

FINE – TUNING MPP (*all2all*)
u_0 u_0 ••• u_0 u_0 u_0 MPP → $\bar{u}_1$ ≈ u_1
N
u_0 u_0 ••• u_0 u_0 u_1 MPP → $\bar{u}_2$ ≈ u_2
u_i u_i ••• u_{j-3} u_{j-2} u_{j-1} MPP → $\bar{u}_j$ ≈ u_j
FINE – TUNING MPP (*steady*)
f f ••• f f f MPP → $\bar{u}$ ≈ u
INFERENCE
① u_0 u_0 ••• u_0 u_0 u_0 MPP → $\bar{u}_1$ ≈ u_1
② u_0 u_0 ••• u_0 u_0 $\bar{u}_1$ MPP → $\bar{u}_2$ ≈ u_2
③ u_0 u_0 ••• u_0 $\bar{u}_1$ $\bar{u}_2$ MPP → $\bar{u}_3$ ≈ u_3
Ⓣ u_0 u_0 ••• $\bar{u}_{T-3}$ $\bar{u}_{T-2}$ $\bar{u}_{T-1}$ MPP → $\bar{u}_T$ ≈ u_T
context size T^S

Figure 6: Schematic representation of MPP *all2all* and *steady* finetuning and inference strategies.

D Results

D.1 Performance on Downstream Tasks

We evaluate all models on the *median relative L^1 error* at a certain snapshot in time on each solution function of interest. For vectorized functions such as the velocity in fluid flow, we evaluate it over the entire vector. Since all our downstream tasks have different solution functions of interest, we provide an overview over the actual functions of interest in Table 6; should there be a list, we compute the mean over all metrics. It also depicts, how the rollout in time was done for each task i.e., either the solution at final time is computed directly with the final time as lead time or autoregressively (AR) as presented in Main Text (8). Within AR, POSEIDON models were always evaluated with uniform (homogeneous) rollout whereas in the case of CNO and CNO-FM, autoregressive (AR) rollout is heterogeneous; for MPP, it is always uniformly autoregressive.

Evaluation of Downstream Tasks with Scaling Plots. In Figures 7 to 21, we present the test errors (y-axis) for all the models (POSEIDON-L, POSEIDON-B, CNO-FM, MPP, FNO on the left sub-figure and POSEIDON-L, POSEIDON-B, CNO, scOT and FNO on the right sub-figure of each figure) vs. the total number of trajectories (time-dependent PDEs) or total number of samples (for time-dependent PDEs) on the x-axis. As mentioned in the Main Text, the POSEIDON models clearly outperform all the baselines on most of the tasks as the corresponding test errors are significantly lower than the baselines for the same number of samples. Note that the metrics **EG** and **AG** (11) were computed based on these plots. We do not include the POSEIDON-T results as they would further clutter the scaling plots. However, the **EG** and **AG** metrics for POSEIDON-T are presented in Table 8.

On Scaling Laws. If we denote the number of trajectories (samples) by M, we can fit power laws of the form,

$$\mathcal{E}_{\text{model}}(M) \approx C_{\text{model}} M^{-\alpha_{\text{model}}}, \tag{81}$$

to the scaling plots in Figure 7 to 21. Here, C_{model} denotes the model-specific scaling factor and the scaling exponent is α_{model}. The scaling exponents, resulting from these fits are presented in Table 7. We observe from this table, that all the models that we consider obey *scaling laws* of the form (81), with different scaling exponents for different problems (MPP-B does not converge in some cases). These include the POSEIDON foundation models which show consistent scaling laws. For instance, POSEIDON-L has a scaling exponent of approximately 0.5 or higher in all the cases except for CE-RM (where all models converge very slowly). Nevertheless, we would like to emphasize that the scaling exponent alone does not govern the final error, except in the asymptotic infinite data limit. Rather, the scaling factor C_{model} in (81) plays a decisive role in determining errors in the pre-asymptotic limited data regime that all downstream tasks correspond to.

Moreover, a closer analysis of the scaling plots reveals a more nuanced picture for the POSEIDON models. In some of the downstream tasks, for instance the Poisson-Gauss benchmark, we see from the scaling plot Figure 20 that both POSEIDON-L and POSEIDON-B display a *biphasic* behavior, with a scaling law of the form,

$$\mathcal{E}_{\text{model}}(M) \approx \begin{cases} C^{w}_{\text{model}} M^{-\alpha^{w}_{\text{model}}}, & \text{if} \quad M \leq M^{pt}_{\text{model}}, \\ C^{\ell}_{\text{model}} M^{-\alpha^{\ell}_{\text{model}}}, & \text{if} \quad M \geq M^{pt}_{\text{model}}, \end{cases} \tag{82}$$

with $\alpha^w < \alpha^\ell$. Thus, the scaling behavior is characterized by two phases, with different exponents. For instance, for the Poisson-Gauss benchmark (Figure 20), we find that $M^{pt} = 32$ for both models. Moreover, for POSEIDON-B, $\alpha_w = 0.23$ and $\alpha_\ell = 0.99$ and for POSEIDON-L, $\alpha_w = 0.33$ and $\alpha_\ell = 0.94$. We speculate that these *phase transitions* separate two phases, a *warmup* phase where POSEIDON is slowly learning about an operator that is very different from those encountered in the pretraining dataset (as is the case with Poisson-Gauss) and a *learning* phase, where fast learning takes place and the model is able to quickly learn the specifics of the downstream task.

Summarizing Downstream Task Performance. In Table 9, we provide a statistical summary of the performance of all models on all downstream tasks by presenting the (median) **EG** and the (mean) **AG** over all tasks. These statistics provide an (average) account of model performance over all downstream tasks and clearly quantify how the POSEIDON family of foundation models significantly outperforms all the baselines.

Table 6: The evaluation metrics are computed for each downstream task on different functions of interest, and rollout is done differently.

Downstream Task	Functions of Interest	Rollout
NS-PwC	(u_x, u_y)	AR
NS-SVS	(u_x, u_y)	AR
NS-BB	(u_x, u_y)	AR
NS-SL	(u_x, u_y)	AR
NS-Tracer-PwC	(u_x, u_y), c	AR
FNS-KF	(u_x, u_y)	direct
CE-RPUI	ρ, (v_x, y), p	AR
CE-RM	ρ, (v_x, v_y), p	direct
SE-AF	ρ	direct
GCE-RT	ρ, (v_x, v_y), p, ϕ	direct
Wave-Layer	u	direct
Wave-Gauss	u	direct
ACE	u	direct
Poisson-Gauss	u	direct
Helmholtz	u	direct

Table 7: Scaling exponents with a power law fit (81)

Dataset	POSEIDON-B	POSEIDON-L	scOT	CNO-FM	CNO	MPP-B	FNO
NS-PwC	0.36	0.49	0.52	0.43	0.62	0.55	0.34
NS-SVS	0.49	0.48	0.77	0.56	0.55	0.53	0.02
NS-BB	0.32	0.51	0.57	0.47	0.64	0.52	0.40
NS-SL	0.36	0.47	0.48	0.46	0.45	0.45	0.59
NS-Tracer-PwC	0.78	0.66	0.59	0.41	0.56	0.43	0.44
FNS-KF	0.98	0.56	1.04	0.30	0.45	0.29	0.43
CE-RPUI	0.35	0.37	0.43	0.27	0.32	0.05	0.23
CE-RM	0.10	0.11	0.09	0.10	0.11	-0.20	0.11
SE-AF	0.30	0.32	0.27	0.35	0.13	0.31	0.24
GCE-RT	0.53	0.59	0.44	0.47	0.31	0.11	0.43
Wave-Layer	0.57	0.51	0.33	0.40	0.51	0.13	0.43
Wave-Gauss	0.59	0.50	0.45	0.37	0.46	0.07	0.34
ACE	0.74	0.85	0.77	0.72	0.47	0.46	0.88
Poisson-Gauss	0.99	0.94	1.07	0.67	0.50	0.71	0.61
Helmholtz	0.38	0.43	0.68	0.42	0.54	0.27	0.31

Table 8: Efficiency gain (EG) and Accuracy Gain (*AG*) for the Poseidon models on all downstream tasks.

	Pretrained Models						Scratch	
	Poseidon-L		Poseidon-B		Poseidon-T		FNO	
	EG	*AG*	EG	*AG*	EG	*AG*	EG	*AG*
NS-PwC	890.6	*24.7*	1024.0	*19.7*	1024.0	*19.8*	1	*1*
NS-SVS	502.9	*7.3*	518.9	*7.9*	212.0	*6.1*	1	*1*
NS-BB	552.5	*29.3*	816.0	*14.7*	365.0	*19.4*	1	*1*
NS-SL	21.9	*5.5*	19.1	*4.7*	9.7	*3.7*	1	*1*
NS-Tracer-PwC	49.8	*8.7*	20.4	*5.4*	35.1	*6.2*	1	*1*
FNS-KF	62.5	*7.4*	16.1	*4.7*	77.9	*5.9*	1	*1*
CE-RPUI	352.2	*6.5*	370.8	*6.2*	909.7	*5.8*	1	*1*
CE-RM	4.6	*1.2*	3.1	*1.1*	2.8	*1.1*	1	*1*
SE-AF	3.4	*1.2*	2.9	*1.2*	2.4	*1.1*	1	*1*
GCE-RT	5.3	*2.0*	3.2	*1.5*	1.7	*1.2*	1	*1*
Wave-Layer	46.5	*6.1*	24.9	*4.7*	14.5	*3.4*	1	*1*
Wave-Gauss	62.1	*5.6*	29.3	*4.3*	19.5	*3.1*	1	*1*
ACE	17.0	*11.6*	8.7	*6.5*	9.8	*7.2*	1	*1*
Poisson-Gauss	42.5	*20.5*	24.4	*13.0*	18.2	*8.4*	1	*1*
Helmholtz	78.3	*6.1*	64.7	*5.0*	64.7	*4.9*	1	*1*

Table 9: (Median) Efficiency gain (EG) and (Mean) Accuracy Gain (*AG*) over all downstream tasks for all models. We also present $\mathcal{N}$(EG) as the number of tasks for which the EG of the model is greater than 10 and $\mathcal{N}$(AG) as the number of tasks where the AG of the model is greater than 2.

	Median EG	Mean AG	$\mathcal{N}$(EG)	$\mathcal{N}$(AG)
Poseidon-L	**49.8**	**9.58**	**12**	**13**
Poseidon-B	24.4	6.71	11	12
Poseidon-T	19.5	6.49	10	12
CNO-FM	10.6	2.91	8	10
MPP-B	2.0	1.82	3	6
CNO	4.6	2.61	5	6
scOT	5.4	2.57	4	8

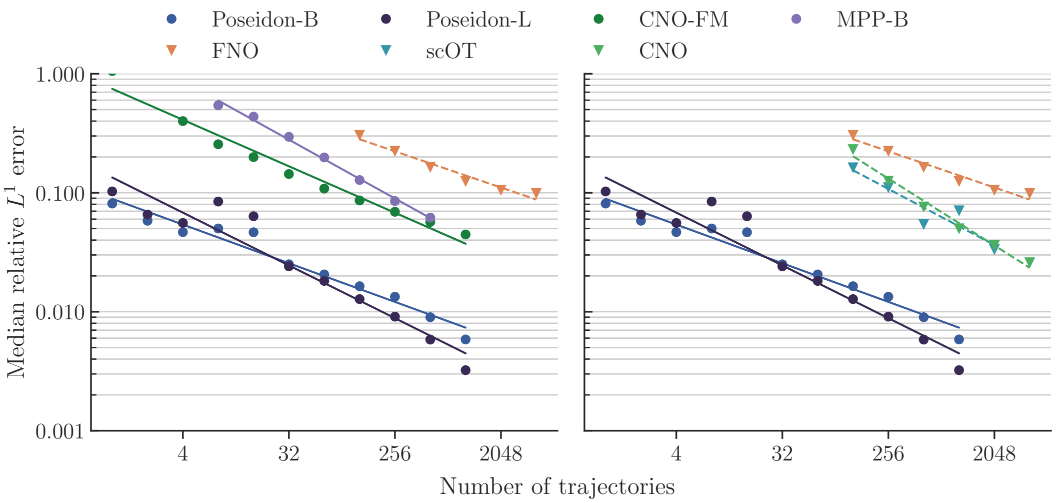


Figure 7: NS-PwC. Number of trajectories vs. median relative L^1 error on the test set.

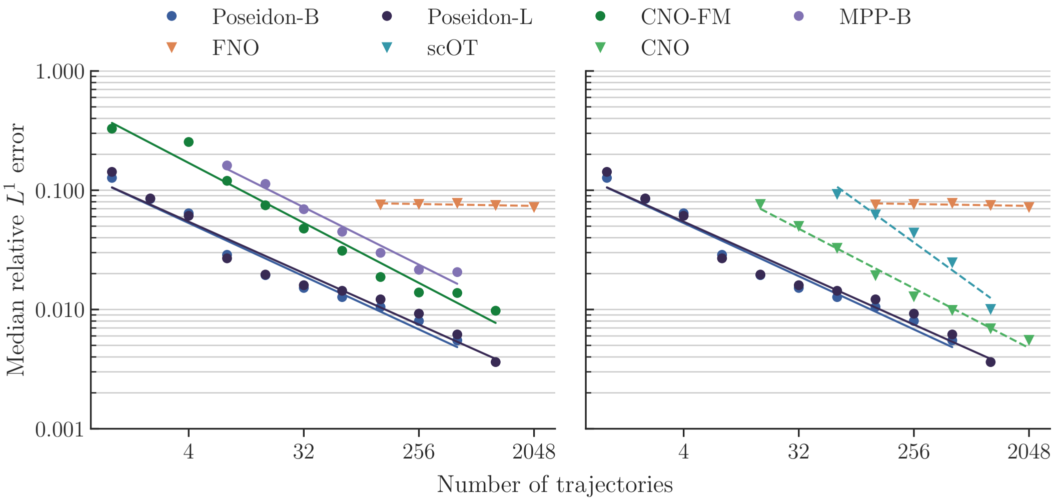


Figure 8: NS-SVS. Number of trajectories vs. median relative L^1 error on the test set.

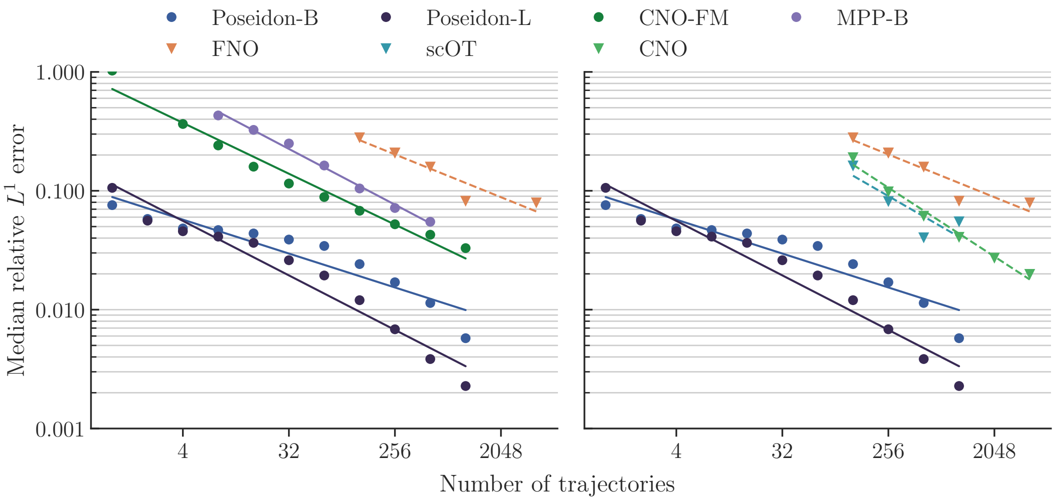


Figure 9: NS-BB. Number of trajectories vs. median relative L^1 error on the test set.

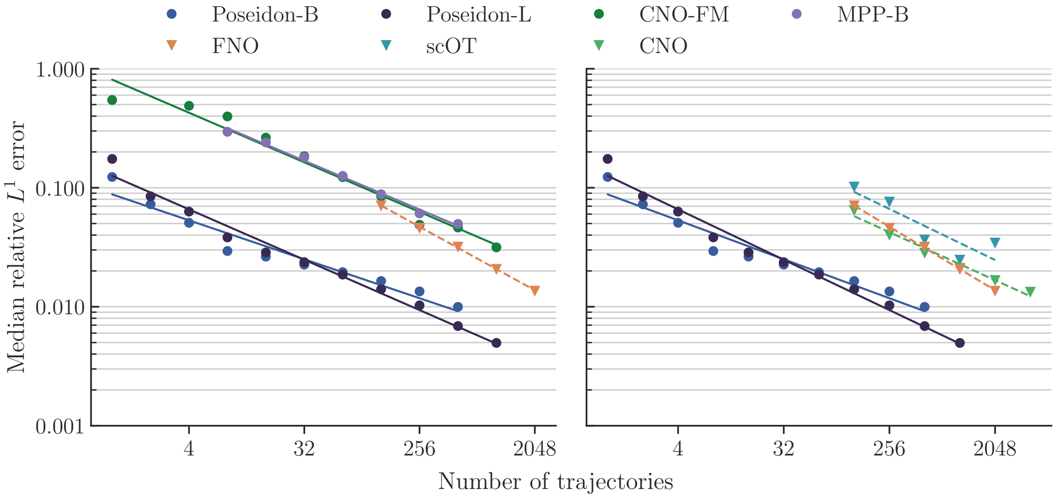


Figure 10: NS-SL. Number of trajectories vs. median relative L^1 error on the test set.

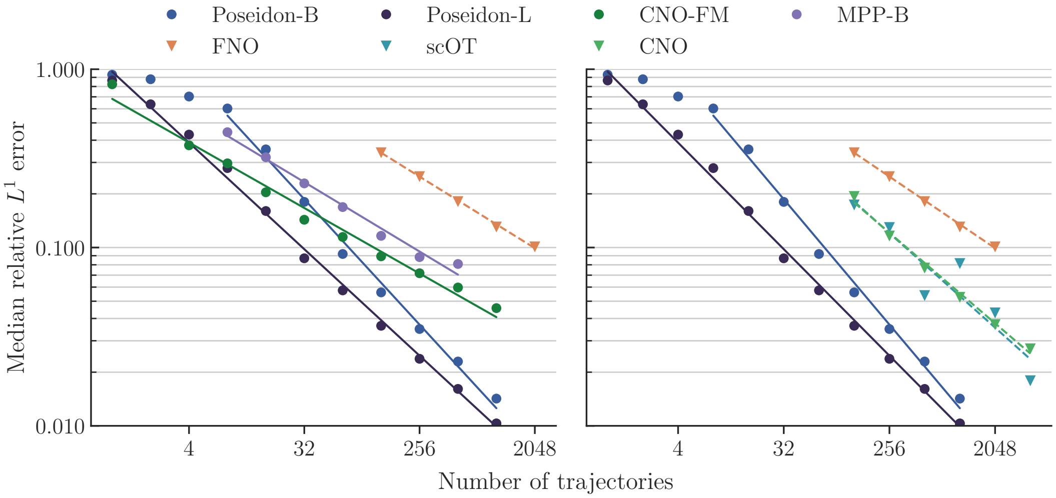


Figure 11: NS-Tracer-PwC. Number of trajectories vs. median relative L^1 error on the test set.

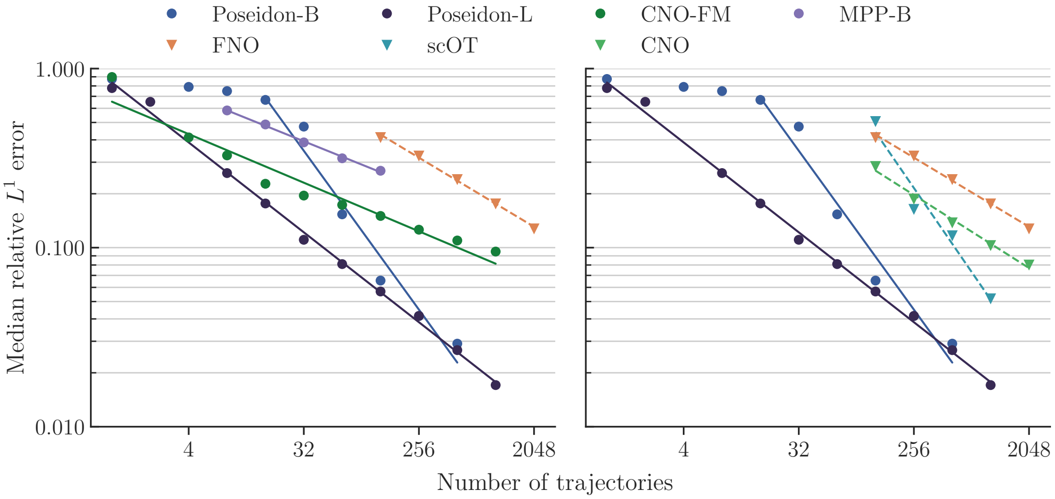


Figure 12: FNS-KF. Number of trajectories vs. median relative L^1 error on the test set.

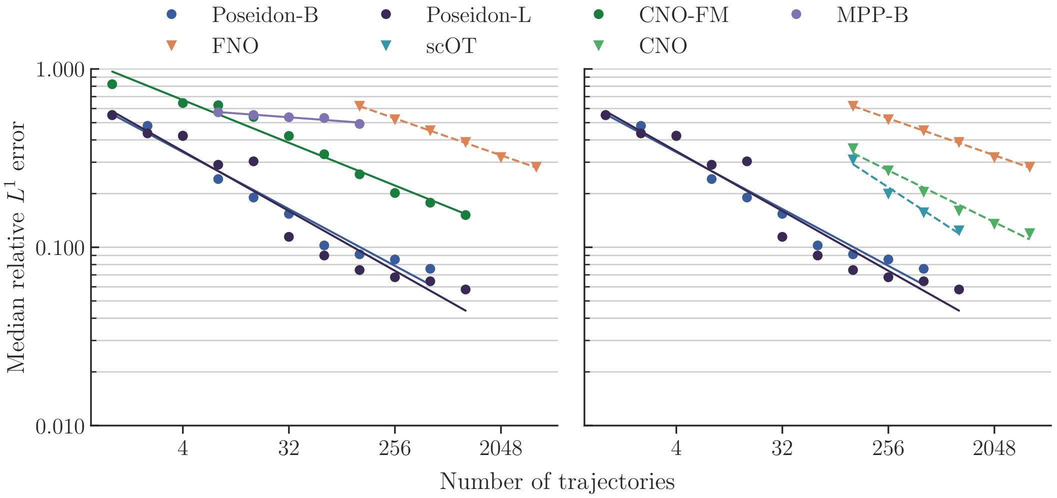


Figure 13: CE-RPUI. Number of trajectories vs. median relative L^1 error on the test set.

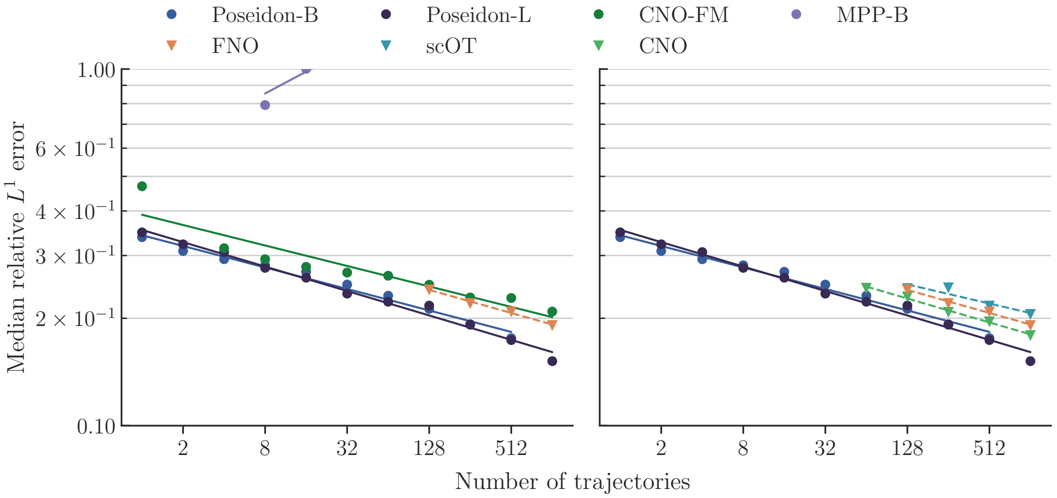


Figure 14: CE-RM. Number of trajectories vs. median relative L^1 error on the test set.

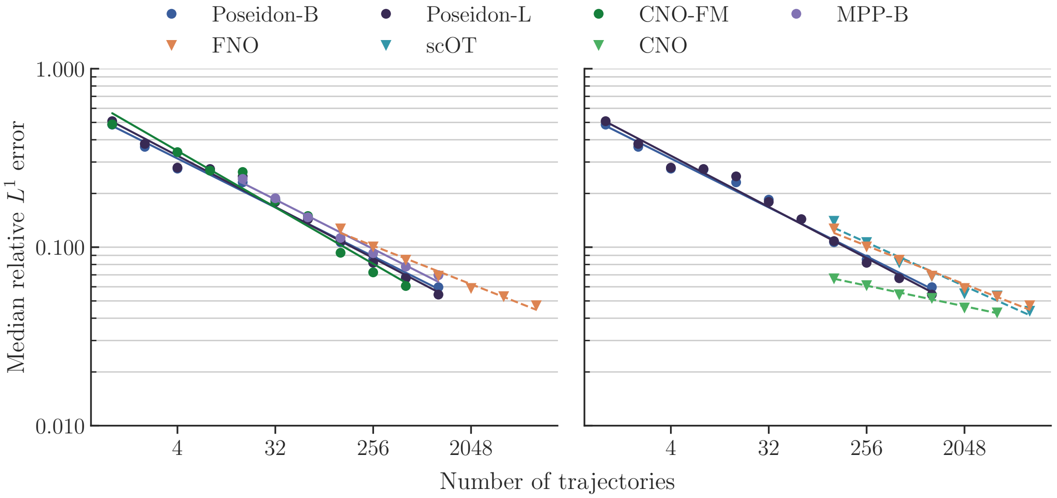


Figure 15: SE-AF. Number of samples vs. median relative L^1 error on the test set.

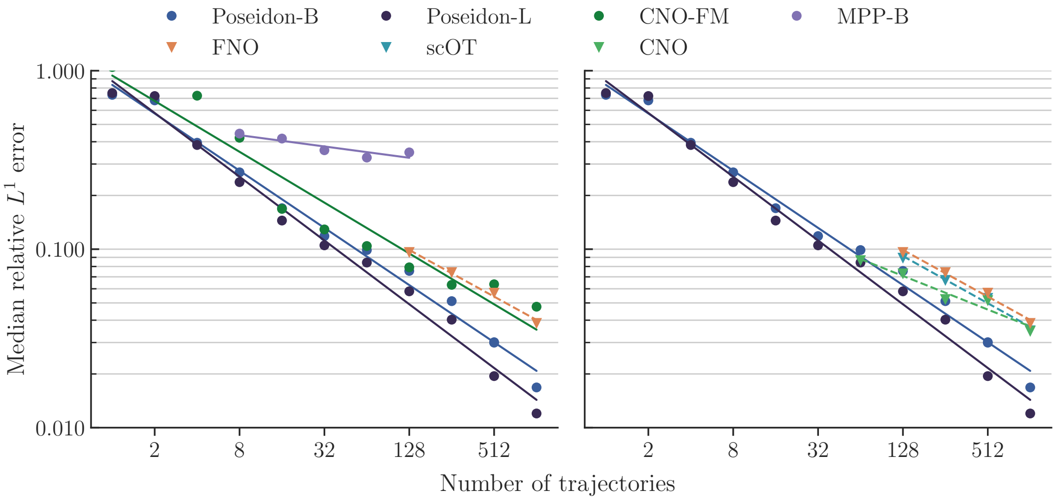


Figure 16: GCE-RT. Number of trajectories vs. median relative L^1 error on the test set.

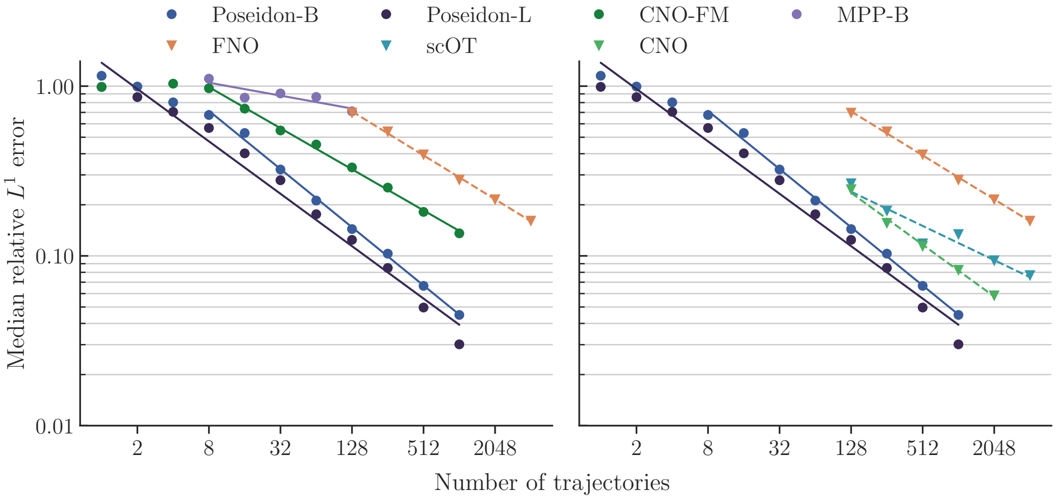


Figure 17: Wave-Layer. Number of trajectories vs. median relative L^1 error on the test set.

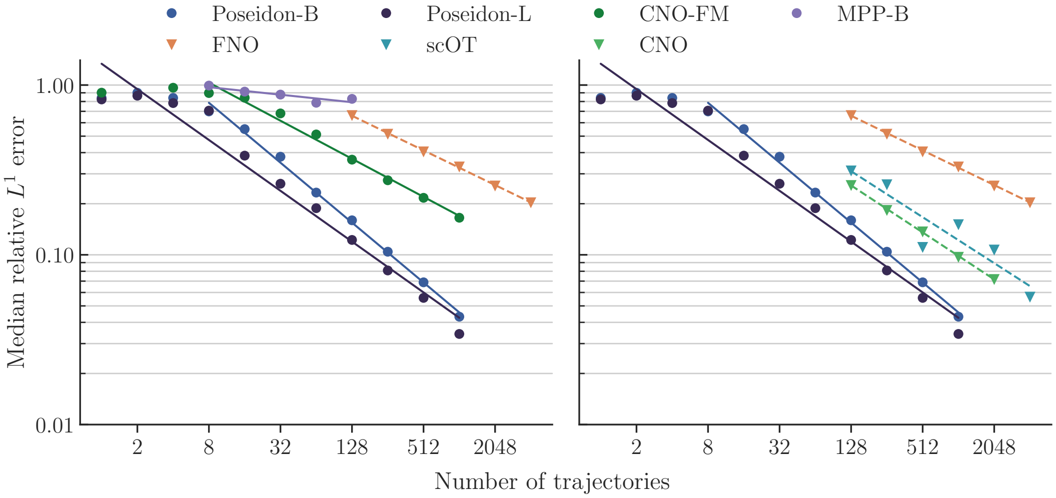


Figure 18: Wave-Gauss. Number of trajectories vs. median relative L^1 error on the test set.

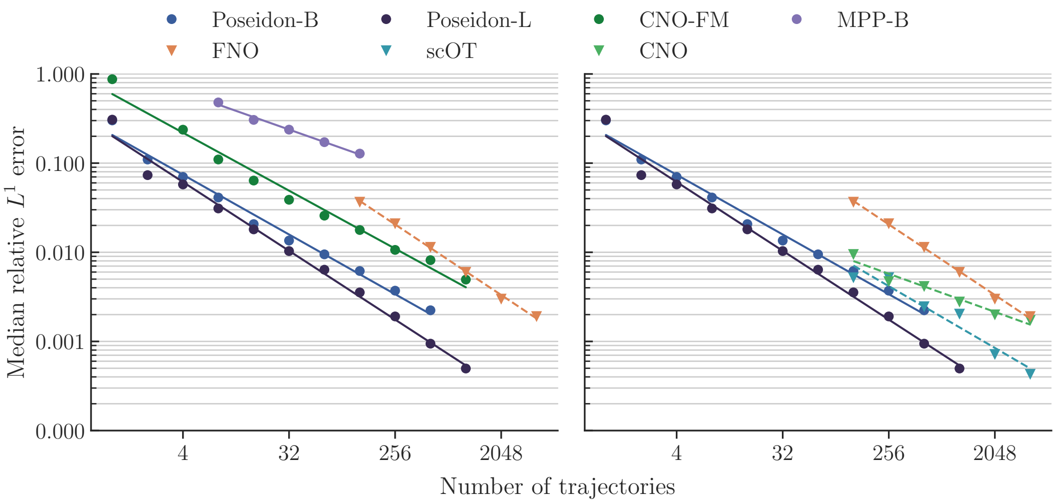


Figure 19: ACE. Number of trajectories vs. median relative L^1 error on the test set.

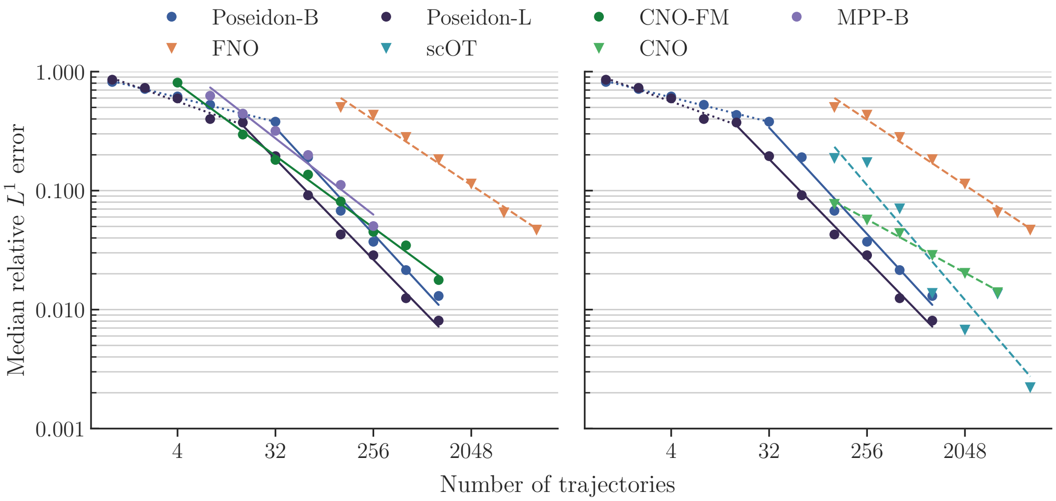


Figure 20: Poisson-Gauss. Number of samples vs. median relative L^1 error on the test set.

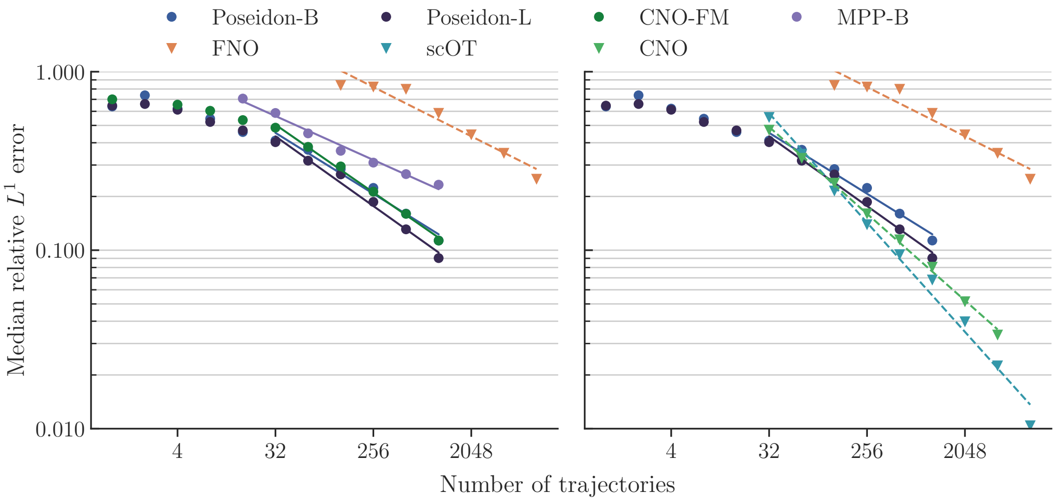


Figure 21: Helmholtz. Number of samples vs. median relative L^1 error on the test set.

D.2 Scaling with respect to Model Size

In Figure 22, we plot how the training loss and evaluation (validation) loss during pretraining changes with model size for the POSEIDON models. We observe from this figure (bottom row) that there is a consistent decay in losses with increasing model size. The role of model size vis a vis downstream tasks has already been shown in the scaling plots of the previous subsection where we compared POSEIDON-L with the smaller POSEIDON-B. The corresponding metrics **EG** and **AG** are shown in Table 8. We also see from the statistical summary Table 9 that there is a gain, on average, in downstream performance with increasing model size for the POSEIDON family of models.

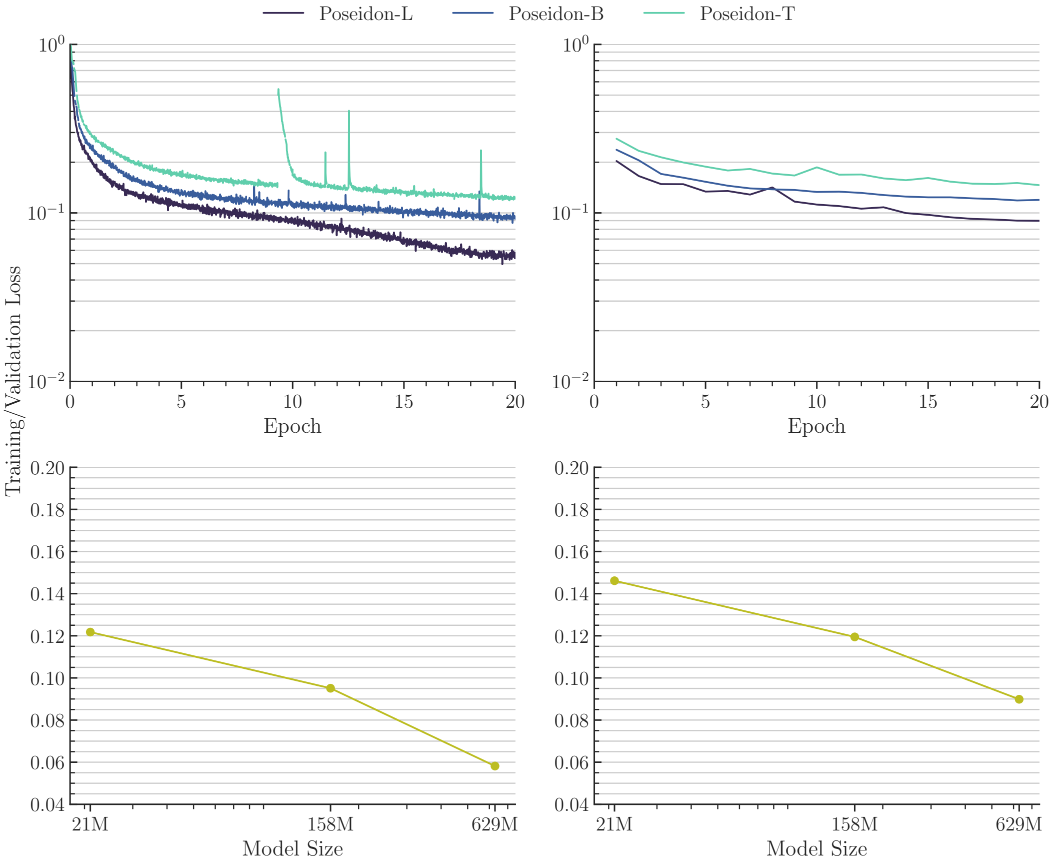


Figure 22: (Top) Training (left) and evaluation (right) losses up to epoch 20 for different model sizes. (Bottom) Scaling at epoch 20 for training loss (left) and evaluation loss (right).

D.3 Scaling with respect to Pretraining Dataset Size and Quality

As mentioned in the Main Text, scaling of the POSEIDON models with dataset size is of great interest. To that end, in Figure 23, we plot the training and evaluation losses, during pretraining, for the POSEIDON-B model, trained with one-eighth, one-half and full size of the pretraining dataset (the details of the corresponding setups are given in Section C). We see from this figure that POSEIDON-B scales with dataset size on the pretraining dataset. Moreover, in Figures 24 to 38 (left subfigure of each figure), we compare the performance of POSEIDON-B, trained on the full pretraining dataset with the same model trained on one-eighth of it.

Moreover, in Figures 24 to 38 (right subfigure of each figure), we compare the performance of POSEIDON-B, trained on the half the pretraining dataset with the same model pretrained on a less diverse dataset (see Main Text and Section C for details of the setup).

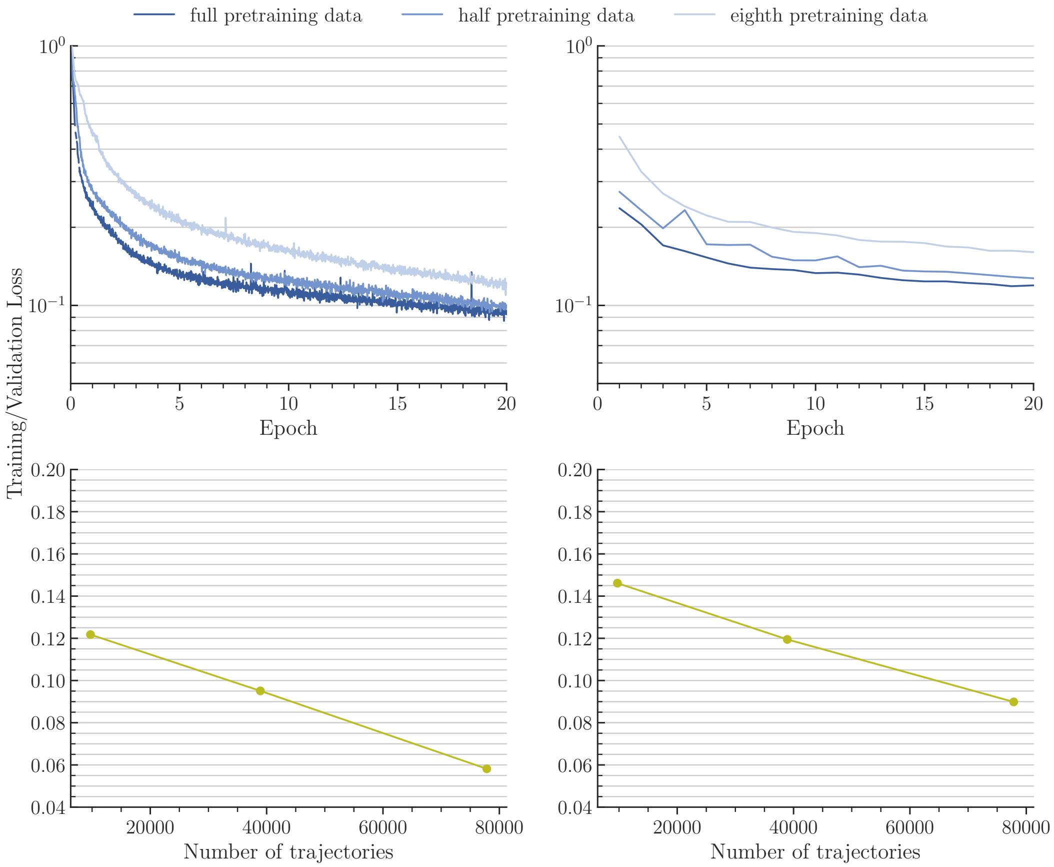


Figure 23: (Top) Training (left) and evaluation (right) losses up to epoch 20 for different pretraining dataset sizes. (Bottom) Scaling at epoch 20 for training loss (left) and evaluation loss (right).

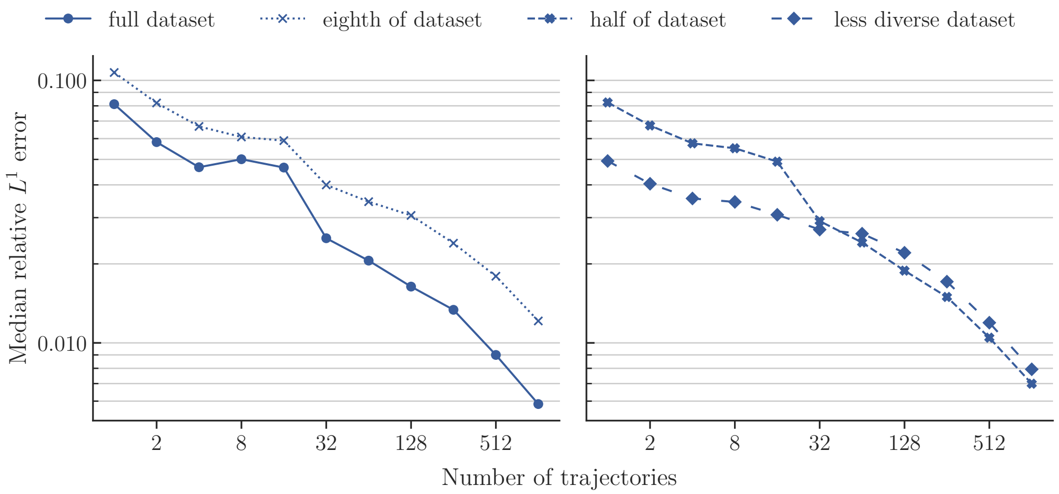


Figure 24: NS-PwC. Number of trajectories vs. median relative L^1 error on the test set.

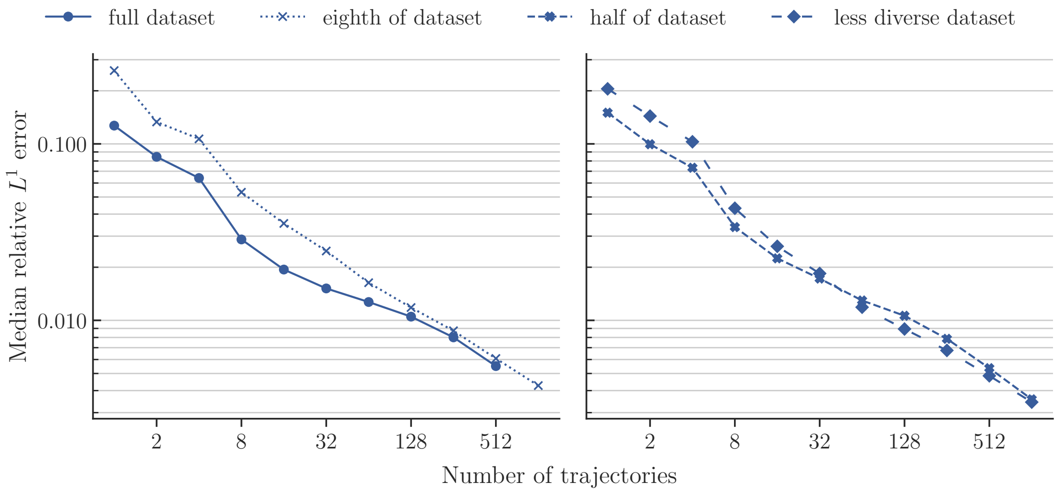


Figure 25: NS-SVS. Number of trajectories vs. median relative L^1 error on the test set.

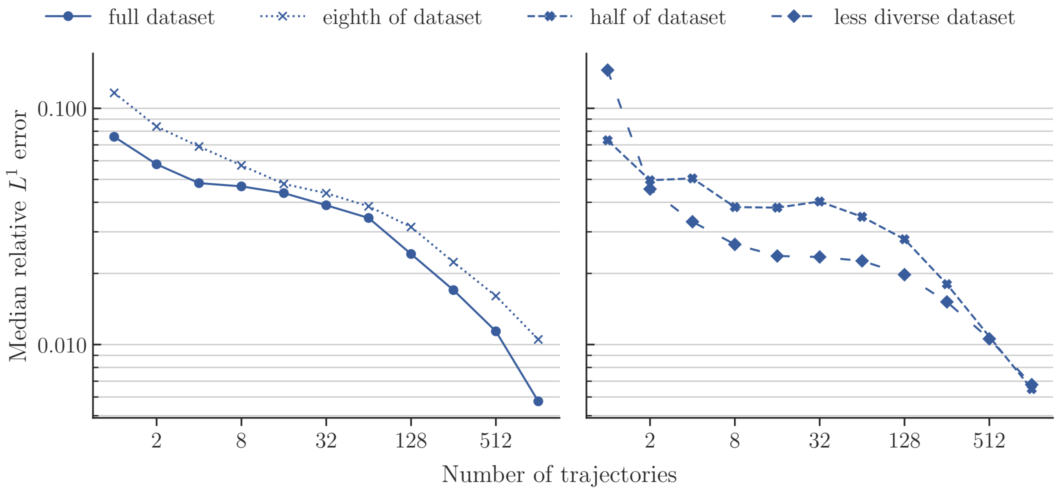


Figure 26: NS-BB. Number of trajectories vs. median relative L^1 error on the test set.

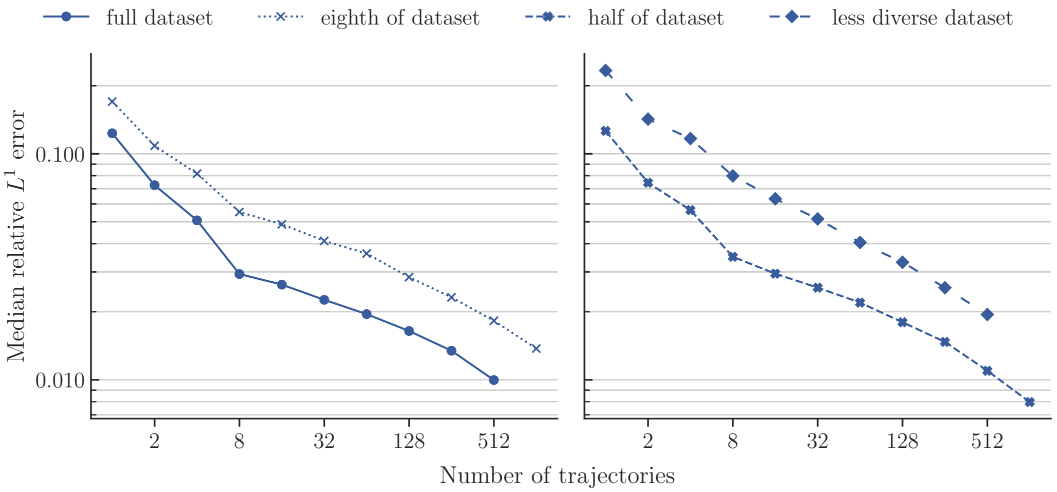


Figure 27: NS-SL. Number of trajectories vs. median relative L^1 error on the test set.

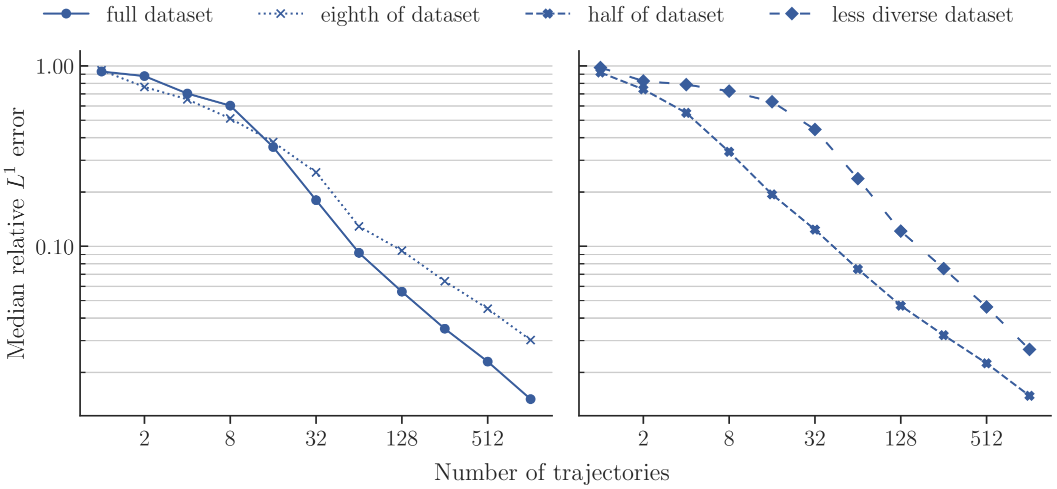


Figure 28: NS-Tracer-PwC. Number of trajectories vs. median relative L^1 error on the test set.

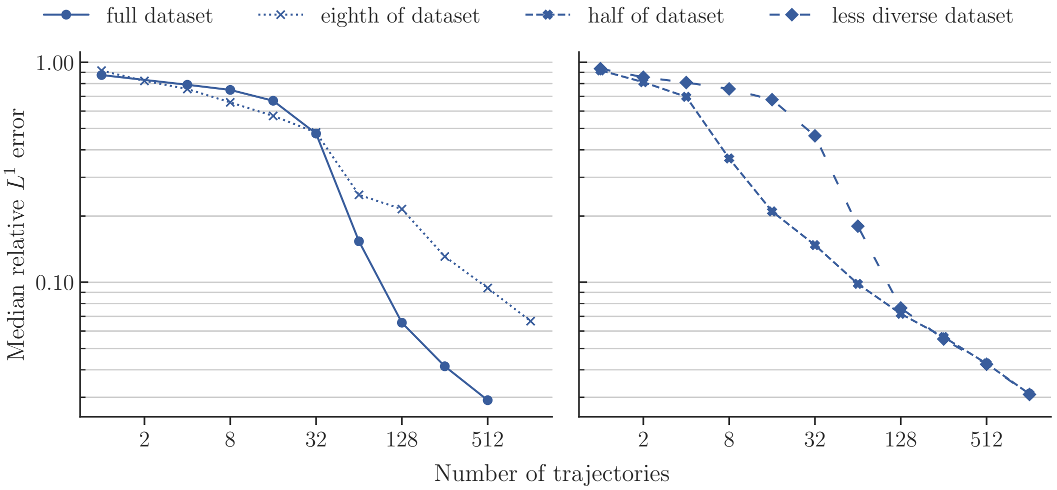


Figure 29: FNS-KF. Number of trajectories vs. median relative L^1 error on the test set.

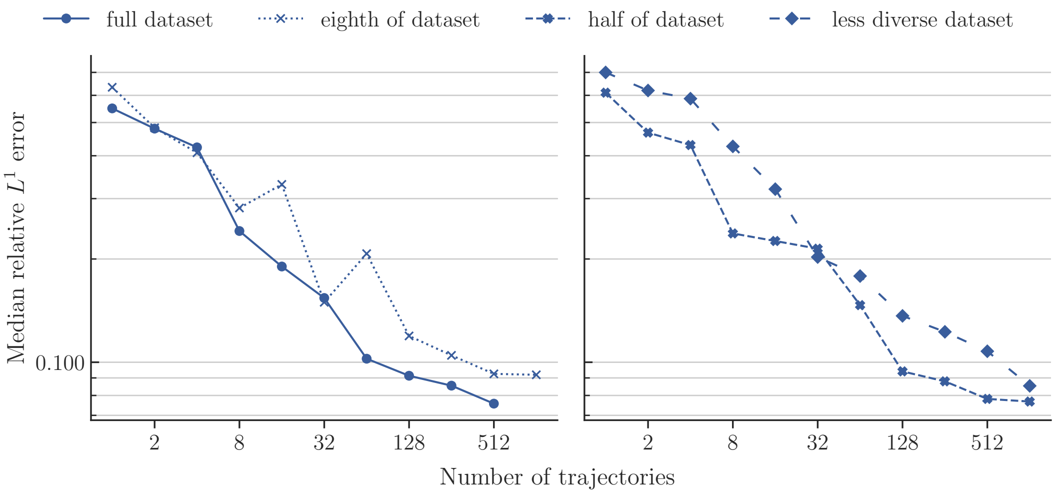


Figure 30: CE-RPUI. Number of trajectories vs. median relative L^1 error on the test set.

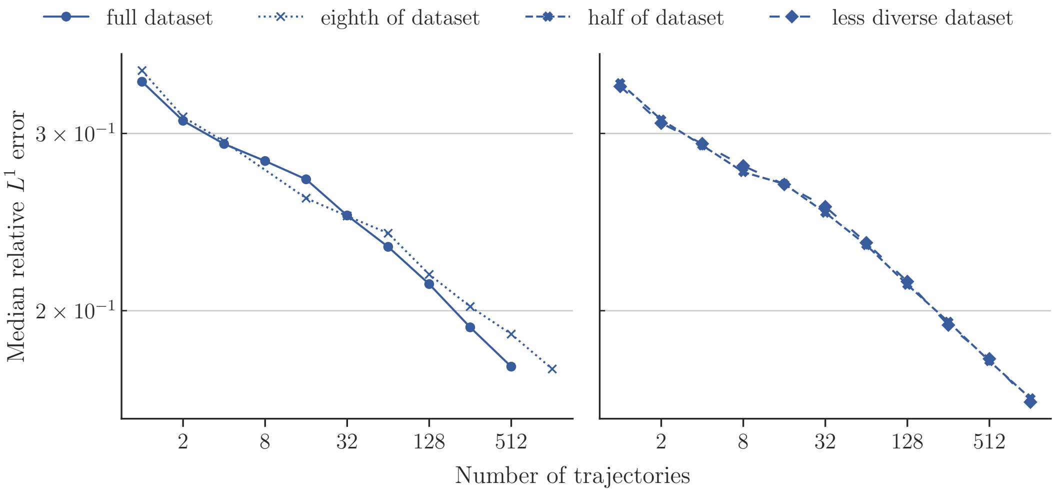


Figure 31: CE-RM. Number of trajectories vs. median relative L^1 error on the test set.

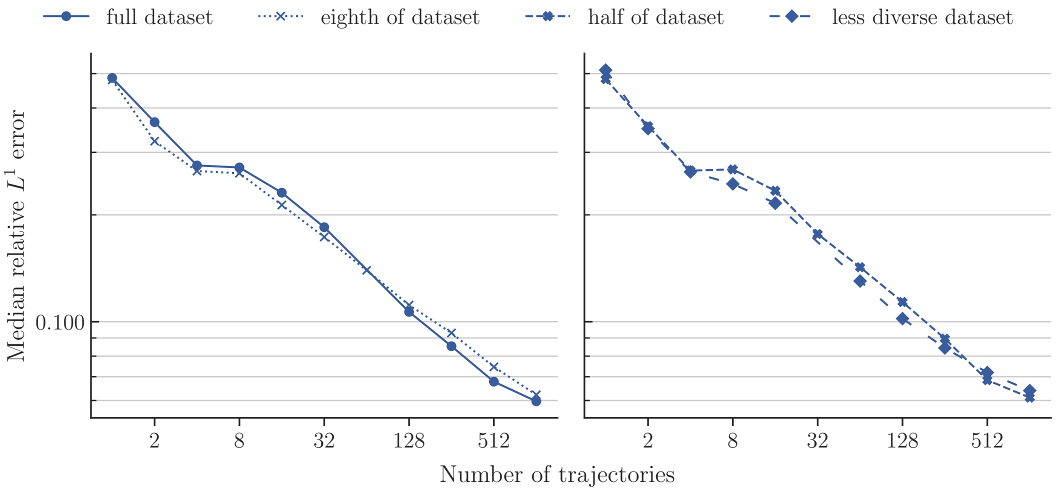


Figure 32: SE-AF. Number of samples vs. median relative L^1 error on the test set.

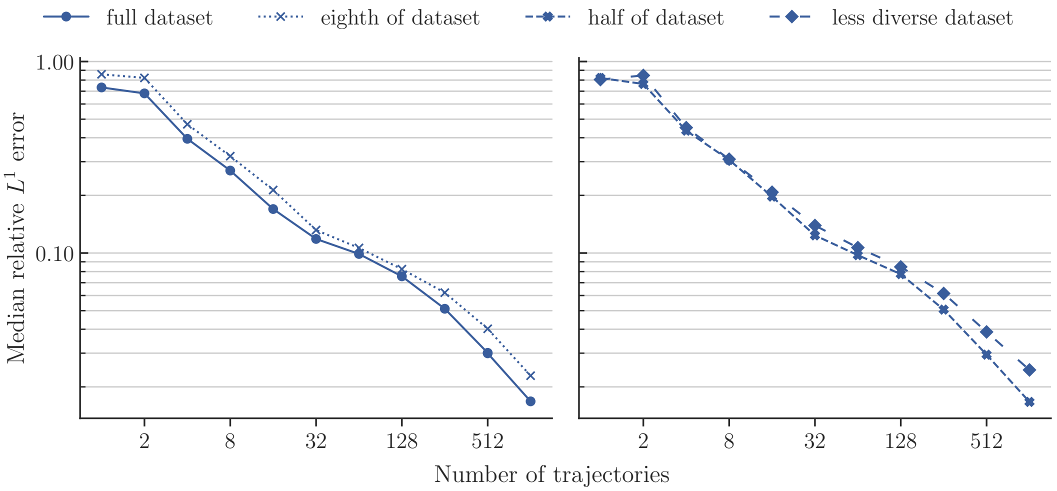


Figure 33: GCE-RT. Number of trajectories vs. median relative L^1 error on the test set.

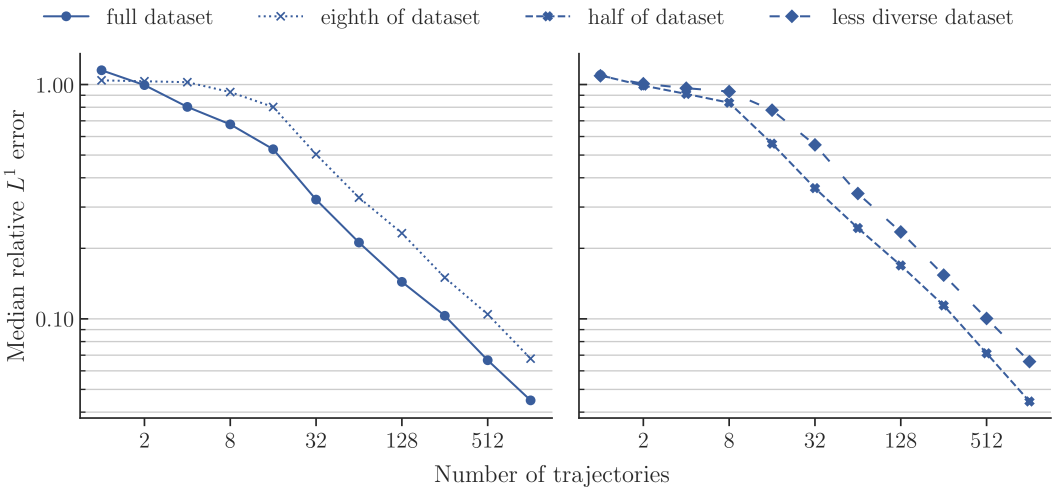


Figure 34: Wave-Layer. Number of trajectories vs. median relative L^1 error on the test set.

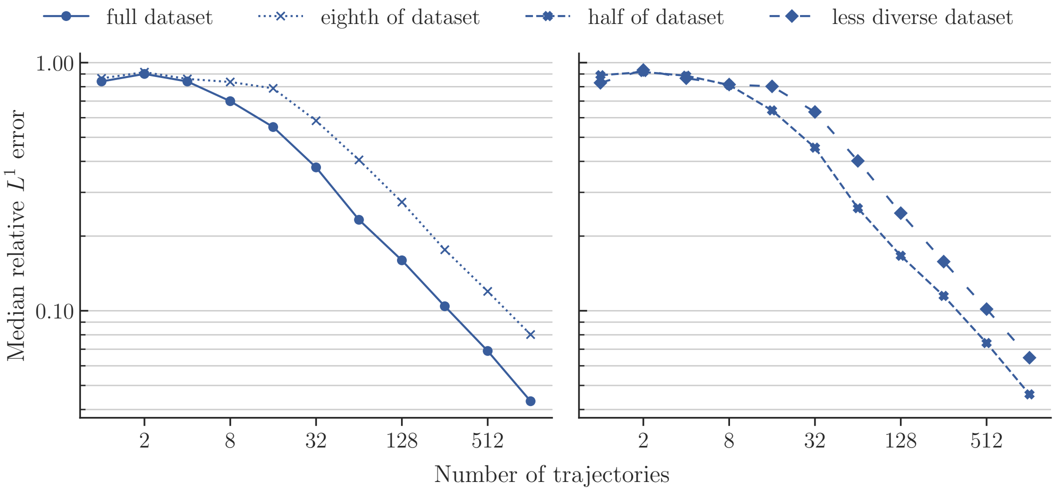


Figure 35: Wave-Gauss. Number of trajectories vs. median relative L^1 error on the test set.

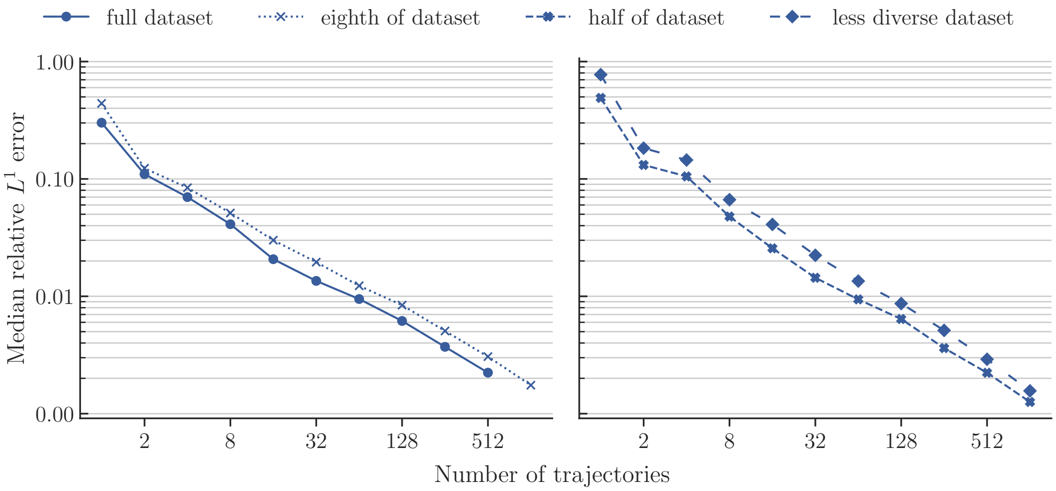


Figure 36: ACE. Number of trajectories vs. median relative L^1 error on the test set.

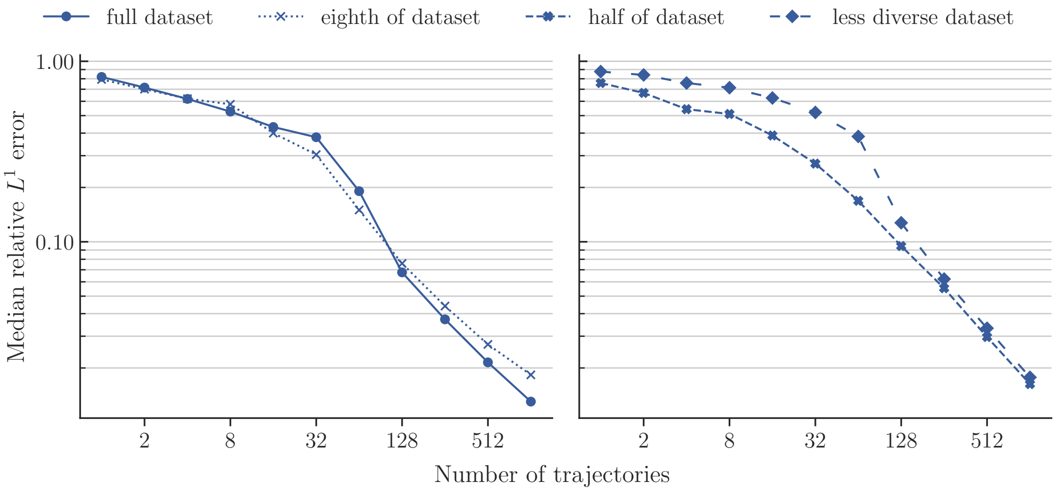


Figure 37: Poisson-Gauss. Number of samples vs. median relative L^1 error on the test set.

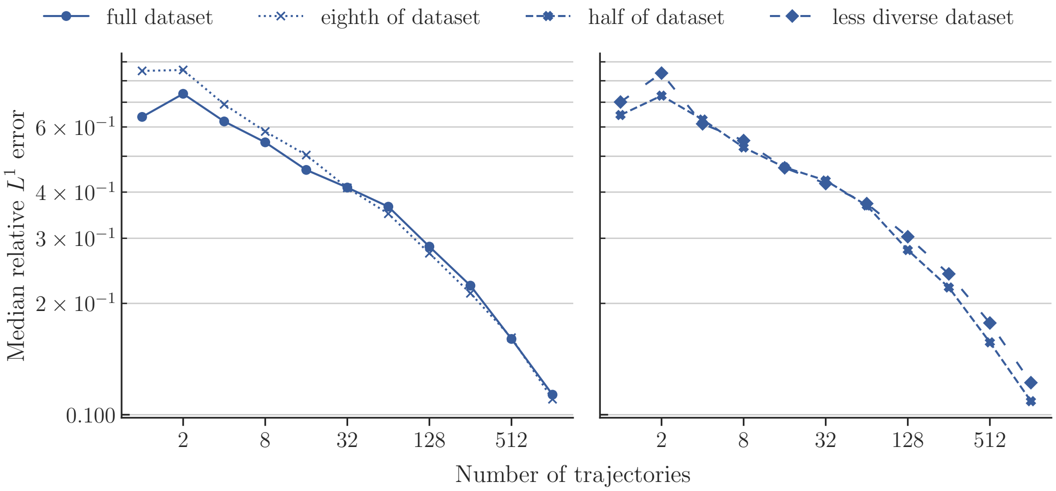


Figure 38: Helmholtz. Number of samples vs. median relative L^1 error on the test set.

D.4 Case Studies

Given the excellent performance of POSEIDON models across the board, including on tasks that involve PDEs (physical processes) not encountered during pretraining, it is important to understand what underpins this performance. To this end, we will present three case studies in order to explain POSEIDON's robust performance.

D.4.1 CE-RPUI

First we consider the CE-RPUI downstream task. Clearly, POSEIDON models perform very well on this task, as shown in Figure 13 as well as Tables 1 and 8. Also, as seen from Figure 67, where we visualize a single random sample for all the variables at time $T = 0.7$, POSEIDON-B is much more accurate, when finetuned on 128 trajectories, than CNO and FNO, which are trained from scratch with the same number of trajectories. Note that the underlying solution is very complex, with a mixture of shocks and roll-up vortices. While POSEIDON captures these shocks and vortices very sharply, CNO and (especially) FNO fail to do so. What explains this impressive performance of POSEIDON on this difficult downstream task?

We start by observing that, like all other downstream tasks, this task is out-of-distribution (o.o.d.) with respect to the pretraining dataset. Although the underlying PDE (compressible Euler equations (37)) is present in the pretraining dataset, this data distribution has not been seen during pretraining. This *o.o.d.* nature of the task is clearly seen from Figure 39, where we plot how the same random sample (visualized in Figure 67)) is inferred with a POSEIDON-B model *zero-shot*. We see from the figure (second column from the left) that the zero-shot results are rather poor. However, even with 1 task-specific example, we see from Figure 39 (third column from left) that at least, some large scale features (such shock locations) are approximated reasonably accurately. With just 4 downstream trajectories, the quality of approximation improves dramatically and even the vortex roll-ups are captured accurately. The quality of approximation continues to improve with 32 and 128 downstream trajectories, as shown in the right-most columns of Figure 39. Thus, from this figure we conclude that a few task-specific samples suffice to accurately approximate the underlying solution operator. This is also evidenced in the scaling plot Figure 13.

How does POSEIDON succeed in learning this complex solution operator with so few samples? We know that the pretraining dataset contains the CE-RP operator, where the initial condition (see Figure 57 for a sample) has a similar four-quadrant Riemann problem structure as the intial conditions in the CE-RPUI benchmark, the main difference being that the interfaces, across which the initial data is discontinuous, are now perturbed sinusoidally, instead of being axis-aligned. However, it is precisely these perturbations that are responsible for the roll-up of small-scale vortices that are absent in the CE-RP operator. Thus, the model potentially needs to learn how to approximate small-scale vortices accurately from some other operator in the pretraining dataset, while learning how to propagate large-scale shocks from the CE-RP operator.

One would think that the CE-KH operator in the pretraining dataset provides the information about vortex roll-ups, see Figure 59 for visualizing a sample. However, the underlying vortices are much larger. So, where does this missing information come from? One possible source could be the CE-CRP operator (see Figure 58) where vortices of many different scales are being formed. Perhaps, the model leverages shock propagation from CE-RP, large vortex roll-ups from CE-KH and small-scale vortex dynamics, as well as curved shock propagation, from CE-CRP in order to provide very good approximation with a few training examples on CE-RPUI. A partial test of this hypothesis is to check if the model, pretrained with the less-diverse dataset that excludes CE-CRP performs worse than the model pretrained with the full dataset. This is already shown in Figure 30 (Right) where the performance of the model, pretrained with the less-diverse dataset is worse than the model trained with the similarly sized but fully diverse dataset. This behavior is further reinforced from Figure 40, where the approximation of the same sample, considered in Figure 39, with these ablated models is shown. As predicted, the model pretrained on the less diverse dataset is clearly less accurate at resolving small-scale vortices than the competing one trained on the more-diverse dataset. It misses the input from the CE-CRP operator regarding small-scale vortex dynamics.

This qualitative analysis illustrates how the POSEIDON model leverages different operators from its pretraining dataset to amortize different aspects in order to construct accurate approximations during

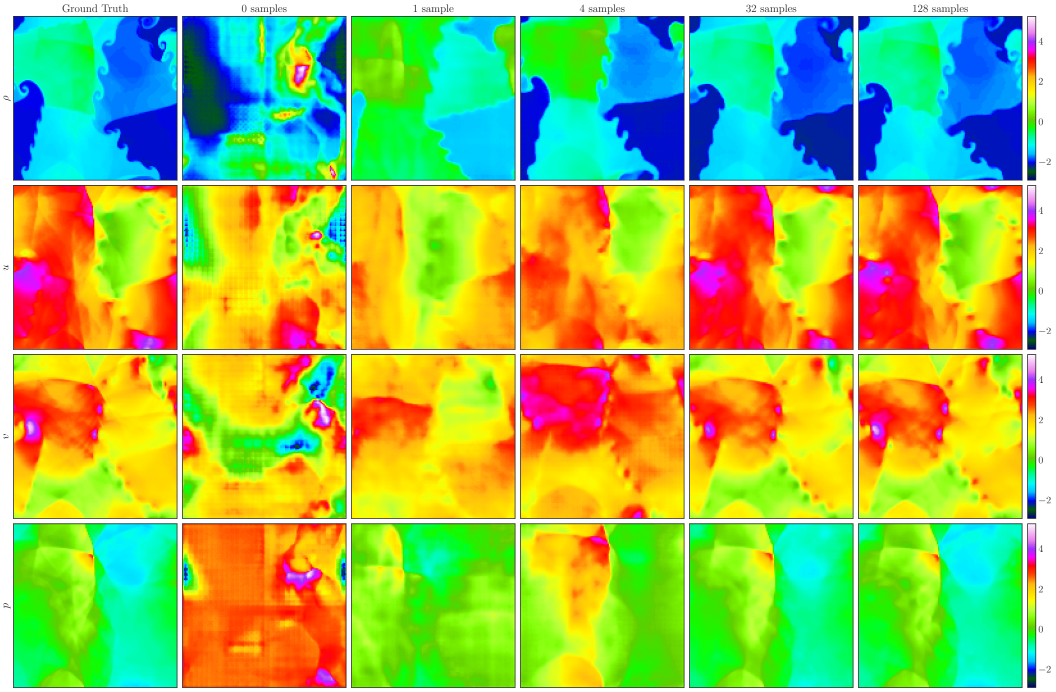


Figure 39: How POSEIDON-B approximates a random sample for the CE-RPUI task when trained with different numbers of task-specific trajectories.

finetuning with a few task-specific examples and throws some light into how a foundation model for PDEs can learn effective representations from its pretraining phase.

D.4.2 ACE

Next, we consider the ACE downstream task, where the underlying PDE is the nonlinear parabolic Allen-Cahn equation (67), which is clearly not included in the pretraining dataset for POSEIDON. More importantly, the type of physics that the Allen-Cahn Equation models is that of *reaction-diffusion*. On the other hand, the PDEs included in the pretraining dataset, Compressible Euler and Incompressible Navier-Stokes at very high Reynolds number, are convection-dominated. Hence, one does not expect that the pretrained model has learned effective representations about reaction-diffusion. Yet, we see from Figure 19 that POSEIDON is very effective at learning this solution operator from a few training examples. This point is also reinforced from Figure 72, where we show how a single randomly chosen sample is well-approximated by POSEIDON. How does POSEIDON learn these *new physics*?

To understand the factors behind POSEIDON's performance, we plot how the same random sample, visualized in Figure 72, is approximated by POSEIDON-B, when fine-tuned with different number of task-specific examples, ranging from 1 to 128, in Figure 41. We observe from this figure that already with just 1 task-specific trajectory, POSEIDON is able to learn the large-scale features of the solution of Allen-Cahn approximately. In particular, it has learnt both front propagation (potentially from all propagating shock waves seen during pretraining) as well as diffusion (spreading) of localized features. With more downstream trajectories, it is able to adjust local features quite well to further approximate the diffuse fronts. This case study shows how POSEIDON can learn new features from a few task-specific training examples.

Given these encouraging results on the ability of POSEIDON to generalize to the unseen physics underlying the Allen-Cahn equation, we investigate this ability further by *freezing the latent space* of POSEIDON during finetuning by setting $\widehat{\theta}_r = \widehat{\theta}_*$, for all r, in the gradient descent procedure (10) for finetuning. Thus, only the *embedding and recovery* parameters are learned and the rest frozen. This results in an *extremely lightweight* model for training as less than 0.5% of the total parameters in POSEIDON are being retrained. Nevertheless, as shown in Figure 42, even this very parsimonious

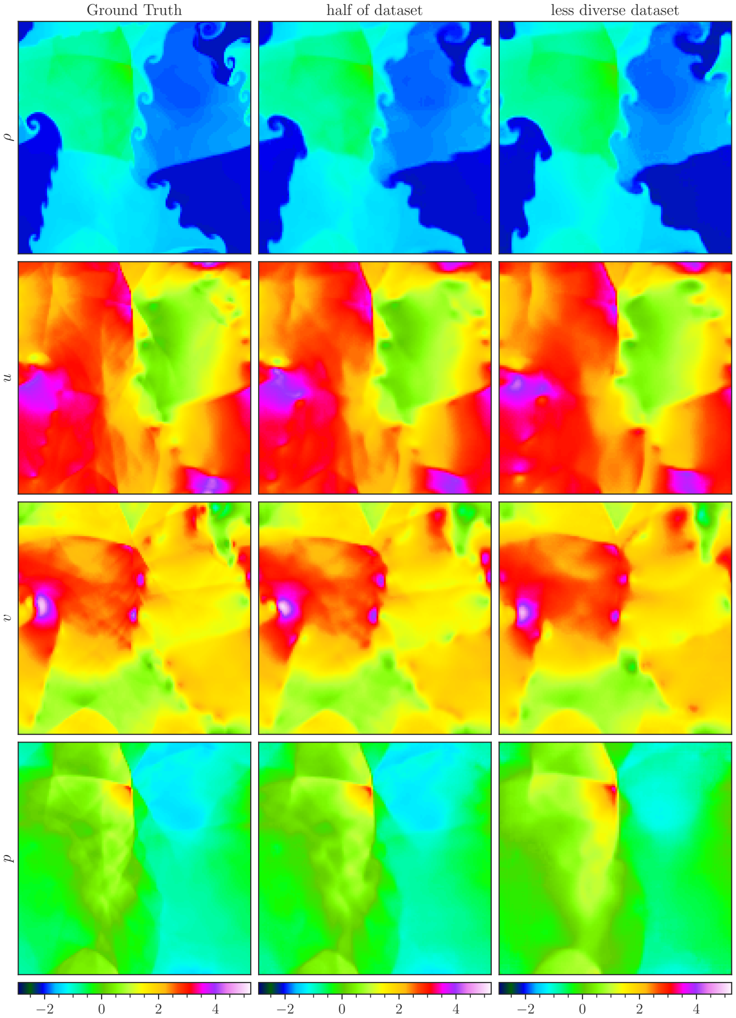


Figure 40: A sample of CE-RPUI when POSEIDON-B is pretrained on half of the pretraining dataset vs. a less diverse pretraining dataset.

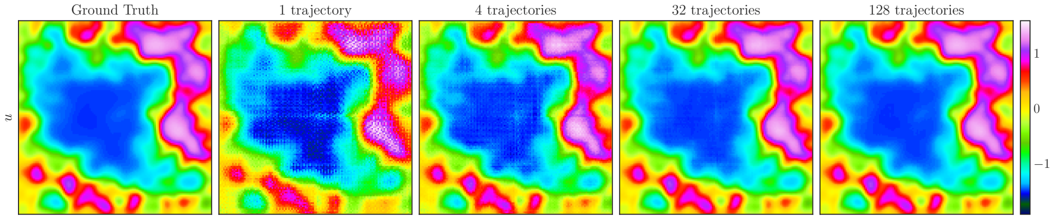


Figure 41: How POSEIDON-B approximates a random sample for the ACE task when trained with different numbers of task-specific trajectories.

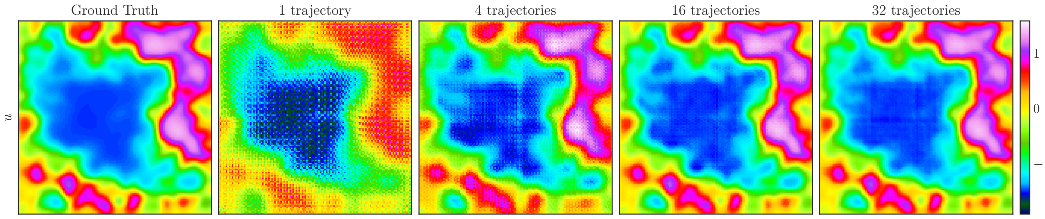


Figure 42: How POSEIDON-B with a *Frozen Latent Representation* approximates the same random sample as in Figure 41 for the ACE task when trained with different numbers of task-specific trajectories.

form of POSEIDON has already learned the solution of the Allen-Cahn equation qualitatively with only one training trajectory, although there is a quantitative mismatch. This mismatch is corrected when further samples are shown to the model. In particular, with 32 trajectories, the error with this model (0.031) is actually lower than FNO with 128 trajectories (0.037) although it is higher than the corresponding model where all the parameters of POSEIDON-B are finetuned (0.014). This experiment demonstrates that the latent representations learned from the equations of fluid dynamics during pretraining are very rich and can unexpectedly contain information about reaction-diffusion equations, which are then leveraged by the *frozen-latent* model to learn the underlying solution operator.

D.4.3 Poisson-Gauss

In our final case study, we consider the Poisson-Gauss task. The underlying PDE is the Poisson equation (68) and the solution operator maps the coefficient, which is a superposition of Gaussians, into the solution. A visualization of the solution operator for a single random sample is shown in Figure 74 and shows how the source is *diffused and smoothed out*.

We remark that this task is very different from the pretraining dataset in various ways. First, the underlying PDE is time-independent in contrast to the two time-dependent PDEs seen during pretraining. Second, the underlying physics of *diffusion* of features and their *smoothing out* is patently different from the physics seen in the pretraining dataset. Finally, the Dirichlet boundary conditions considered here are also different from the periodic boundary conditions of the pretraining dataset. Nevertheless, we see from the scaling plot Figure 20 and Table 1 and 8 that POSEIDON models perform very well in this case. This is also observed from Figure 74, where we observe that POSEIDON-B learns this particular random sample far better than CNO and FNO, with the same number (512) of training samples. To understand the reasons behind POSEIDON's performance, in Figure 43, we again plot how this foundation model approximates this particular random sample, when trained with an increasing number of task-specific samples. We see from this figure that for 1 sample, the approximation is very poor, indicating how much *out-of-distribution* this task is, with reference to the pretraining dataset. In fact, the model simply learns to approximate the input. However, within a few samples (16), it has learnt that the input needs to be spread (diffused) out. It takes about 128 samples for the model to realize that the input needs to be both spread out as well as smoothened and by 512 samples, the local adjustments needed to further smoothen the output have been made.

A few remarks are in order to explain this qualitative picture. First, POSEIDON could have used the first few samples in training to *forget* the information from the pretraining phase. Yet, it does not do that and uses the very first sample to already just output the identity operator. Then, there appears to be a *warmup* phase where the model slowly learns the underlying physics, for instance diffusion and smoothening and then a fast learning phase where the operator can be better approximated. This qualitative picture is also consistent with the observed *biphasic* power scaling, see the subsection on scaling laws in section D.1, and the fact that there is a *phase transition* between the warmup and fast learning phases in the power law (82). This case study sheds further light into how POSEIDON can learn *unseen physics* from a few task-specific training examples.

As with the Allen-Cahn equations of the previous section, we further study the factors underpinning the ability of POSEIDON to generalize to this PDE by *freezing the latent space* of POSEIDON during

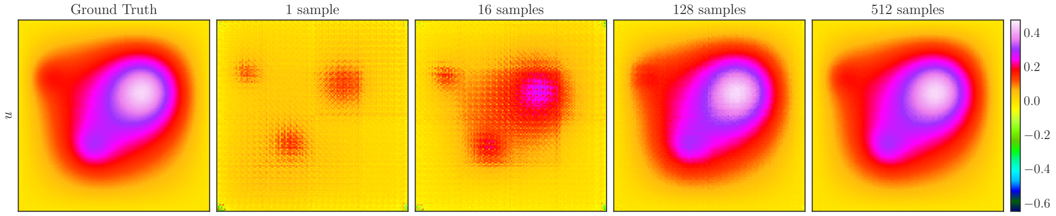


Figure 43: How POSEIDON-B approximates a random sample for the Poisson-Gauss task when trained with different numbers of task-specific samples.

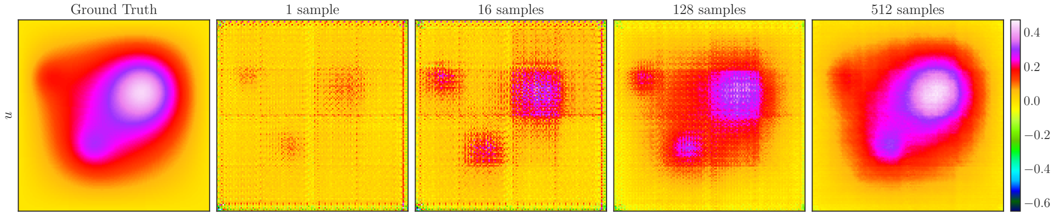


Figure 44: How POSEIDON-B, with a *Frozen Latent* Representation, approximates a random sample for the Poisson-Gauss task when trained with different numbers of task-specific samples.

finetuning by setting $\widehat{\theta}_r = \widehat{\theta}_*$, for all r, in the gradient descent procedure (10) for finetuning. Thus, only the *embedding and recovery* parameters are learned and the rest frozen. As shown in Figure 44, even this frozen-latent form of POSEIDON has already learned the basic features of the underlying solution operator, i.e., Diffusion and Smoothing, qualitatively with only a few training samples, although there is a quantitative mismatch. This mismatch is corrected when further samples are shown to the model. In particular, with 512 trajectories, the error with this model (0.11) is significantly lower than FNO (0.282) although it is higher than the corresponding model where all the parameters of POSEIDON-B are finetuned (0.022). This experiment further demonstrates that the latent representations learned from the equations of fluid dynamics during pretraining are rich enough to even contain information about the a priori unrelated physics of steady state diffusion, which are then leveraged by the *frozen-latent* model to learn the underlying solution operator.

D.5 Results with DPOT

Table 10: Efficiency gain EG ((11) with $S = 1024$ for time-dependent and $S = 4096$ for time-independent PDEs) and Accuracy Gain (*AG*) ((11) with $S = 128$ for time-dependent and $S = 512$ for time-independent PDEs) for DPOT and tested downstream tasks.

	Finetuned DPOT				DPOT from Scratch			
	M		L		M		L	
	EG	*AG*	EG	*AG*	EG	*AG*	EG	*AG*
NS-PwC	44.8	*12.5*	39.7	*12.0*	17.0	*6.1*	23.3	*10.2*
NS-SL	4.5	*2.4*	4.7	*2.4*	2.1	*1.3*	3.0	*1.6*
FNS-KF	0.0	*1.0*	0.0	*0.9*	0.0	*0.8*	0.0	*0.8*
CE-RPUI	53.5	*3.7*	53.6	*3.6*	26.1	*2.5*	31.2	*2.9*
SE-AF	3.5	*1.2*	4.7	*1.4*	4.4	*1.3*	5.1	*1.4*
Wave-Layer	23.5	*5.5*	28.9	*6.0*	14.1	*3.6*	17.8	*4.2*
Wave-Gauss	25.2	*4.4*	27.8	*4.5*	18.0	*3.3*	20.5	*3.6*

The DPOT foundation model [19] has been trained on operators for the compressible and incompressible Navier-Stokes equations, Reaction-Diffusion equations and Shallow-Water equations. The model has been setup to take a sequence of time steps for a time-dependent PDE and output the next time step. However, we can modify it for finetuning for our **OLT** operator learning task by following exactly the same procedure as for finetuning the MPP foundation model. For steady state problems, an identical procedure as with MPP is used. This allows us to perform a fair comparison between DPOT and the POSEIDON models proposed here.

To this end, we consider DPOT-M (with 120 M parameters) and DPOT-L (with 509 M parameters) which are comparable in size to the POSEIDON-B and POSEIDON-L models, respectively. Given compute constraints, we focus this comparison on a representative subset of 7 downstream tasks which are listed in Table 10. Moreover, a trained-from-scratch DPOT model, with the Adaptive FNO architecture, is also employed for each task to evaluate DPOT's model performance. For both finetuning and training models from scratch, we employed the Adam optimizer [25] with a weight decay of 10^{-6}, and a *1cycle* learning rate policy. For finetuning DPOT models, the maximum learning rate was set to 10^{-4}, and training was conducted for 100 epochs. When training models from scratch, we used a maximum learning rate of 10^{-3} and trained for 200 epochs.

The resulting EG and AG scores are presented in Table 10. These scores should be compared with the corresponding EG and AG scores of POSEIDON-L and scOT from Table 1 and POSEIDON-B from Table 8. Comparing these results, we make the following observations,

- For all these tasks except SE-AF, POSEIDON is significantly better, both in terms of efficiency and accuracy gains, to the corresponding DPOT model. Even for SE-AF, the models are very comparable. The superiority in performance of POSEIDON is seen very clearly when we consider the mean AG scores over these 7 downstream tasks which amount to POSEIDON-L (8.14), POSEIDON-B (6.5), DPOT-M (4.39) and DPOT-L (4.4). Hence, the POSEIDON-L model is almost *twice* more accurate than both the DPOT models considered here. In fact, the DPOT models' performance lies in-between CNO-FM with an average AG score of 2.66 and the POSEIDON models. Similar results also hold for the efficiency gain score.
- Surprisingly, DPOT foundation models do not seem to scale with model size, at least on this set of 7 representative downstream tasks as seen from the mean AG scores of 4.4 for both the DPOT-M and DPOT-L models where an increase of the number of parameters by a factor of 5 does not lead to any noticeable increase in model performance on downstream tasks.
- As surprisingly, the stand-alone DPOT neural operators performed well on this dataset. For instance, the average AG score of trained-from-scratch DPOT-L is 3.53, which is only 25% lower than the DPOT-L foundation model. On the other hand, POSEIDON-L is almost 5

times more accurate than the underlying scOT neural operator. These results indicate that DPOT foundation models do not harness latent representations as well as POSEIDON does and they rely on the capacity of the underlying neural operator to learn downstream tasks.

Taken together, our results indicate that (our modification of) DPOT performs better than the CNO-FM and MPP foundation models but is significantly inferior to the POSEIDON models. Moreover, the lack of scaling with model size for DPOT on downstream tasks and questions over how it uses latent representations further point to the advantages of POSEIDON over this competing model. Nevertheless, this comparison merits further study.

D.6 Further Ablations and Results

D.6.1 On all2all training

The all2all training strategy, described in the Main Text, aims to leverage the semi-group structure of the solution operator of the time-dependent PDE (1) to scale-up the training data per trajectory. As shown in Figure 2 (d), we use every possible pair of snapshots, per trajectory, in the learning process leading to the loss function (6). It is instructive to compare this strategy with the *vanilla* training strategy based on the loss function (5). As this strategy is applicable for any (time-dependent) operator learning algorithm, we study it for the CNO model [60] here. To this end, we consider the NS-SL task and compare the all2all and vanilla strategies and plot the results in Figure 45 to observe that the all2all training strategy significantly outperforms the vanilla training strategy for this task.

However, there is a caveat with the all2all strategy. It lies in the computational cost of training as the number of training pairs grows *quadratically* with the number of available time snapshots at which the trajectory is sampled. One option to reduce this cost is to *select a subset of snapshots* from within all available snapshots per trajectory and apply all2all training to this subset, bringing down the computational cost proportionately by the relative reduction in the cardinality of the selected subset. Yet, there is the possibility that by sampling too few snapshots, the overall error will increase.

To investigate this trade-off, we consider the NS-PwC task and the CNO model. The data for this task is available in the time-interval [0,0.7], sampled at 14 time snapshots (excluding the initial time 0). Denoting the ith-snapshot by t_i with $i = 0, 1, \dots, 14$, We select the following subsets of time snapshots,

$\mathcal{T}_{14}$: Snapshots at $t_0 = 0$ and $t_{14} = 0.7$. The training only considers learning the map between initial datum and solution at final time t_{14}. Samples corresponding to identity function are also included.

$\mathcal{T}_7$: Snapshots at t_0, t_7, t_{14}

$\mathcal{T}_2$: Snapshots at $t_0, t_2, t_4, t_6, t_8, t_{10}, t_{12}, t_{14}$

$\mathcal{T}_1$: Snapshots at t_j, for all $0 \leq j \leq 14$

For each of the above subsets of time snapshots, all2all training is used leading to $3, 6, 36$ and 120 training pairs per trajectory for $\mathcal{T}_{14}, \mathcal{T}_7, \mathcal{T}_2, \mathcal{T}_1$, respectively.

In Figure 46, we plot the test error vs. number of trajectories. From the left panel of this figure, we see that there is consistent gain in accuracy as a more dense sampling of the snapshots is performed. The models are monotonically more accurate as we go from $\mathcal{T}_{14}$ through $\mathcal{T}_7$ to $\mathcal{T}_2$. However, we also observe from Figure 46 that going beyond $\mathcal{T}_2$ to $\mathcal{T}_1$ does not yield any further decrease in test error as the difference between the newly added snapshots and the existing ones in $\mathcal{T}_2$ is not statistically significant enough to aid the training process. Moreover, by choosing $\mathcal{T}_2$ over $\mathcal{T}_1$, we reduce the computational cost of training by a factor of 3.3. These considerations motivate us to a not too dense sampling strategy for pretraining (and finetuning) our foundation models.

D.6.2 Direct. vs. Autoregressive Inference

As mentioned in the Main Text, our time-conditioned models can either be directly evaluated at the time of interest, or an autoregressive rollout can be performed (see Equation 8 of the Main Text). This can per se be of any form that the user wants, i.e. with homogeneous step-sizes in time, or with heterogeneous step-sizes in time. For simplicity, we only consider homogeneous autoregressive rollouts for POSEIDON, scOT and FNO models, for the CNO models we find a slight performance boost with a heterogeneous rollout strategy.

Figure 47 shows for the NS-PwC and the Wave-Layer downstream task how the error behaves when using direct or (homogeneous) autoregressive rollouts. We can directly see that it depends very much on the task at hand, as autoregressive rollout works better for the NS-PwC task, whereas direct lead-time input works better for Wave-Layer; this seems to be very dataset- and dynamics-dependent. We therefore choose the best strategy for each task which is listed in Table 6.

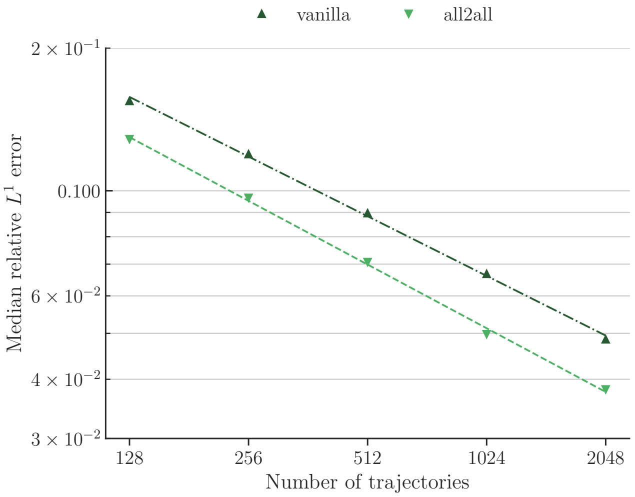


Figure 45: NS-SL. Testing errors of the CNO models trained in an *all2all* and *vanilla* manner. Performance improves with all2all training.

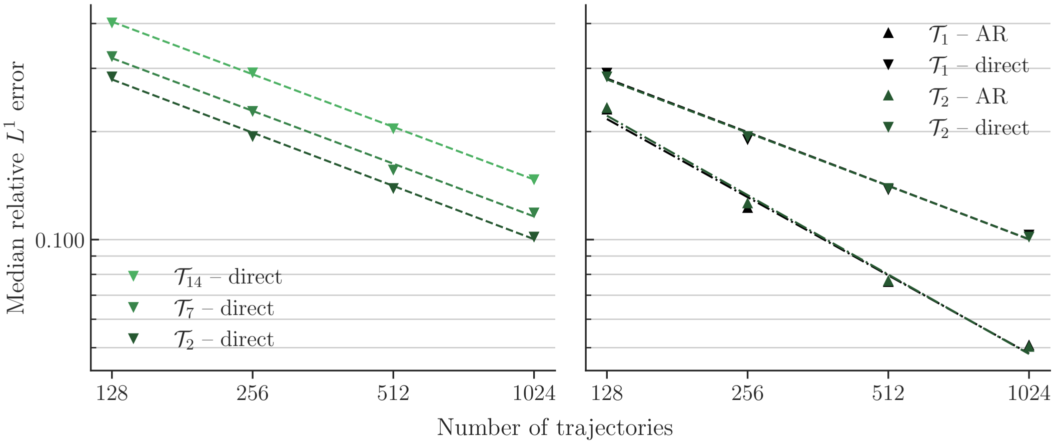


Figure 46: NS-PwC. Testing errors of the CNO models trained in an *all2all* manner on different $\mathcal{T}_i$ trajectories. (Left) Errors from directly evaluating the trained models. Performance improves as denser trajectories are incorporated. (Right) Saturation effect observed. Adding denser trajectories no longer enhances performance, as the additional samples are statistically less significant.

D.6.3 Error Growth over Time for POSEIDON-B

Autoregressive inference can only work better than direct lead-time input when the error that accumulates at every step is smaller than the error obtained by direct lead time input. In Figure 48, we can directly see that the error scales *sub-linearly* for the NS-PwC experiment and this is true in general for our downstream tasks. This leads to two observations. First, there is no blow-up (for instance exponential growth) of error in time with these models. Second, the fact that the error grows in time proves that it is harder to predict the solution at final time from initial data than predicting time-averaged quantities. In other words, the L^∞-error in time will be greater than the L^1-error. This justifies our choice of evaluating different models at the final lead time of the underlying task.

To further demonstrate how POSEIDON compares with FNO over time, we plot errors for the NS-PwC and NS-SL experiments as a function of time with both models in Figure 49. We observe from this figure that the difference in error between FNO and POSEIDON-B actually grows over time and is the highest at the final time as FNO has much larger rate of error growth over time than POSEIDON, justifying our decision to evaluate models with respect to error at the final time.

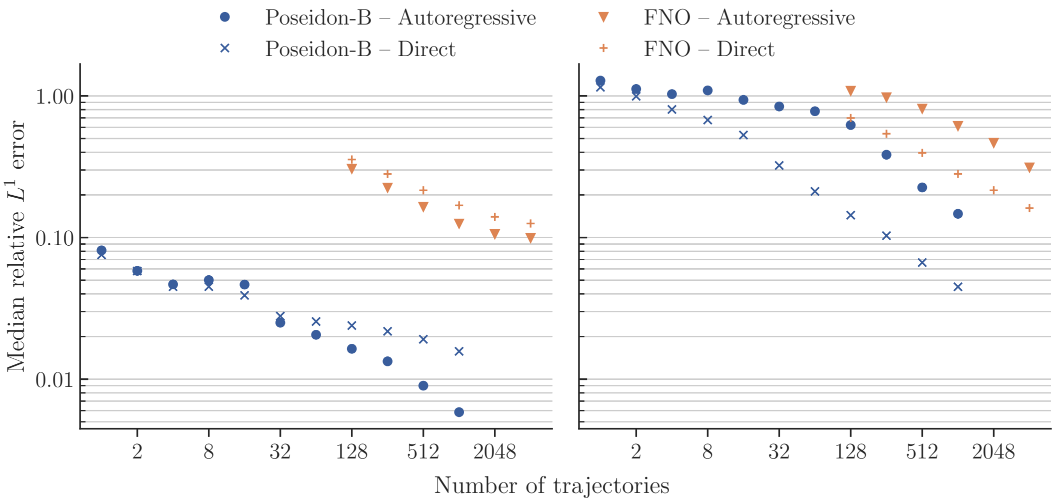


Figure 47: Homogenenous autoregressive rollout vs. direct lead-time input on NS-PwC (left) and Wave-Layer (right).

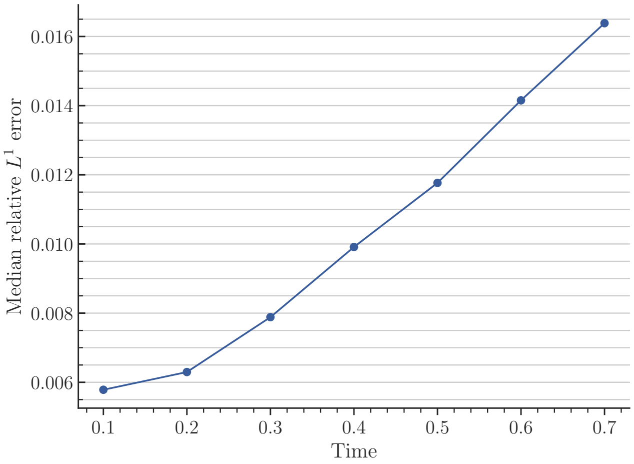


Figure 48: Error accumulation for autoregressive rollout of POSEIDON-B finetuned on 128 trajectories of the NS-PwC dataset.

D.6.4 Out-of-distribution Time Extrapolation

Here, we consider the NS-SL downstream task. As mentioned before, FNO (and other neural operators) were trained from scratch as well as POSEIDON (and other foundation models) were finetuned to learn the solution up to a final lead time of $T = 0.7$. We want to investigate how the POSEIDON foundation model and the neural operator baseline (relatively) perform when we consider an *out-of-distribution time extrapolation* at the downstream task level. To this end, in Figure 50, we plot the test errors, with respect to increasing number of task-specific trajectories, for both FNO and POSEIDON-B, but evaluated at final times of $T = 0.7$ and the extrapolated final time of $T = 1.0$. A homogeneous autoregressive rollout is used in all cases. We observe from this figure that both POSEIDON-B and FNO are worse at extrapolating in time than they are at predicting within the time-period that they have been trained on. In addition to significantly outperforming FNO at both time $T = 0.7$ and at the extrapolated time of $T = 1.0$, POSEIDON-B in fact performs relatively better at out-of-distribution than FNO. It is best seen from the **EG** metric (11), where POSEIDON's **EG** ≈ 20 for time $T = 0.7$ is improved to **EG** ≈ 30 for time $T = 1.0$. This gain can be attributed to the fact that during pretraining, POSEIDON models have been trained for a longer time horizon.

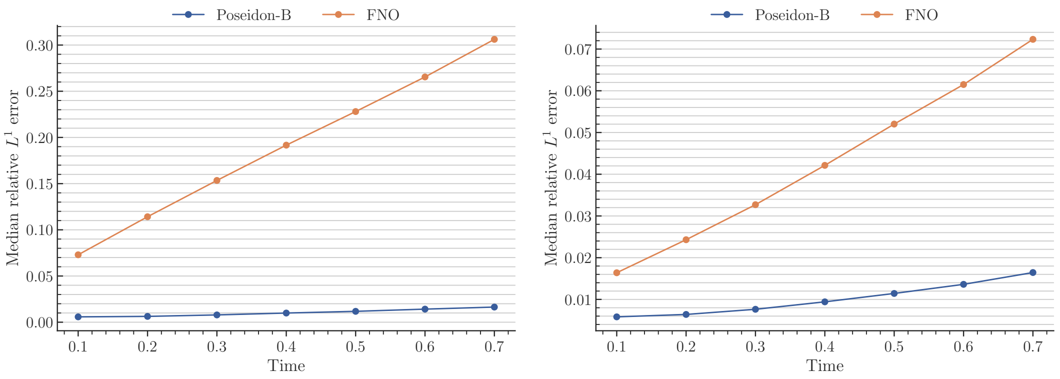


Figure 49: Error accumulation for the finetuned Poseidon-B and FNO for 128 training trajectories on NS-PwC (left) and NS-SL (right).

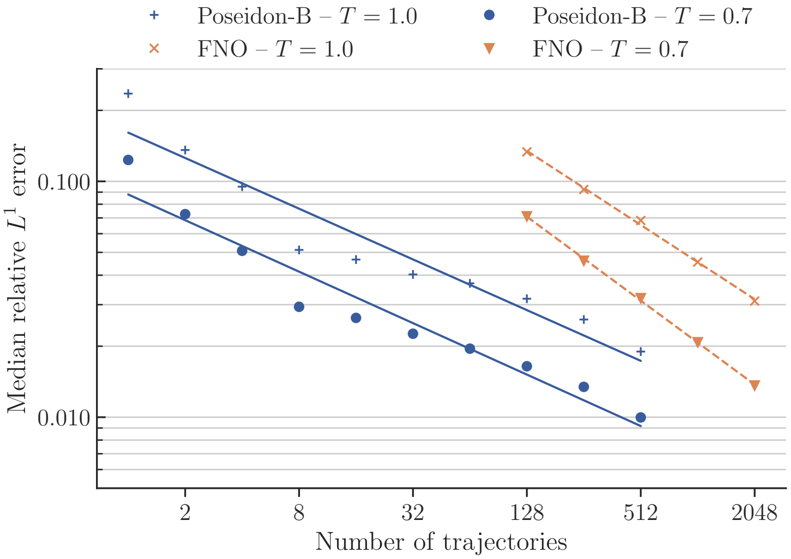


Figure 50: Out-of-distribution extrapolation in time for Poseidon-B and FNO on NS-SL up to $T = 1$.

D.6.5 Generalization of Poseidon with respect to Changing PDE Parameters

Several of our downstream tasks such as GCE-RT, Wave-Layer, Wave-Gauss and Helmholtz involve operators that map the coefficient in the PDE to its solution. This setup is very different from the pretraining dataset where the underlying solution operators only map the initial data to solutions at later times and there is no PDE coefficient that is encountered. Nevertheless, from Tables 1 and 8, we observe that the Poseidon models generalize very well to these very different setups for the operators for downstream tasks. To further test the ability of Poseidon to generalize for different PDE parameters, we consider the Navier-Stokes Equations ((31)) with a viscosity coefficient $\nu = 4 \times 10^{-3}$. The ground truth data is generated using the Azeban spectral hyper viscosity solver [62]. This new viscosity coefficient is very different from the setup of the pretraining data and downstream tasks considered so far as in all of them, only a hyperviscosity of 4×10^{-4} was applied to high-enough Fourier modes in order to model the incompressible Euler equations with zero viscosity. In this *new* task, the initial conditions are identical to the NS-PwC downstream task. We see from Figure 51 that Poseidon-B generalizes very well to this new viscosity coefficient and outperforms FNO readily, in terms of both sample efficiency and accuracy. In particular, the AG and EG scores of Poseidon-B are $EG = 925.5$ and $AG = 47.5$, which are completely comparable to (even better than) the scores of $EG = 1024$ and $AG = 19.7$ (see Table 8 for the original NS-PwC task). Taken

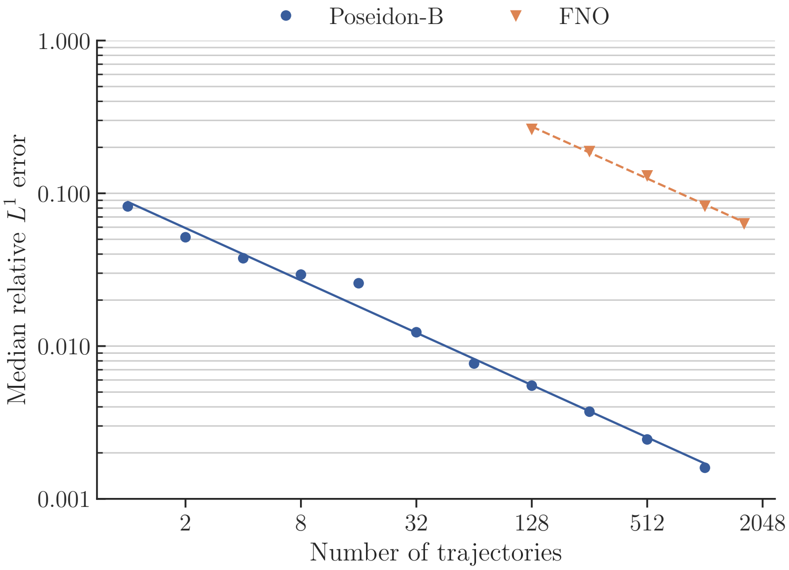


Figure 51: Error for the NS-PwC downstream task, but with viscosity $\nu = 4 \times 10^{-3}$ (on all modes) instead of 4×10^{-4} applied only on high-enough Fourier modes to simulate the inviscid limit

together with other downstream tasks involving different PDE coefficients, this experiment clearly demonstrates the ability of POSEIDON to generalize to different PDE parameters via finetuning.

D.6.6 POSEIDON Evaluated on Different Grids

As POSEIDON is based on an operator transformer (scOT), it can be evaluted on grid resolutions, different from the underlying computational grid. Following [3], we can simply downsample (upsample) the input function from the given grid to the computational grid, process the input with POSEIDON and upsample (downsample) the output from the computational grid to the given grid resolution. We perform this evaluation of POSEIDON-B on multiple grid resolutions for the NS-PwC task and present the result in Figure 52 to observe that the test error is (approximately) invariant to the grid resolution.

D.6.7 Robustness of Poseidon with respect to Noise

To study how robust POSEIDON is to noise, we consider the downstream CE-RPUI task and at inference time, we add Gaussian noise to the inputs (initial conditions) at different noise-to-signal ratios (NSRs) of 0.1%, 1% and 3% respectively. The resulting errors, computed with respect to a Ground Truth where the outputs are not noisy, for varying numbers of training trajectories, are shown in Figure 53. The errors in the zero noise (clean) case are also shown in this Figure. We observe from this figure that POSEIDON-L's performance is robust to input noise and the error does not grow significantly even when the noise level is an appreciable 3%, demonstrating the robustness of this foundation model with respect to noise.

D.6.8 Histograms of Errors for Different Tasks

In Figure 54, we plot the distribution of errors across the test set for all downstream tasks with the POSEIDON-B model, finetuned with 128 trajectories (samples).

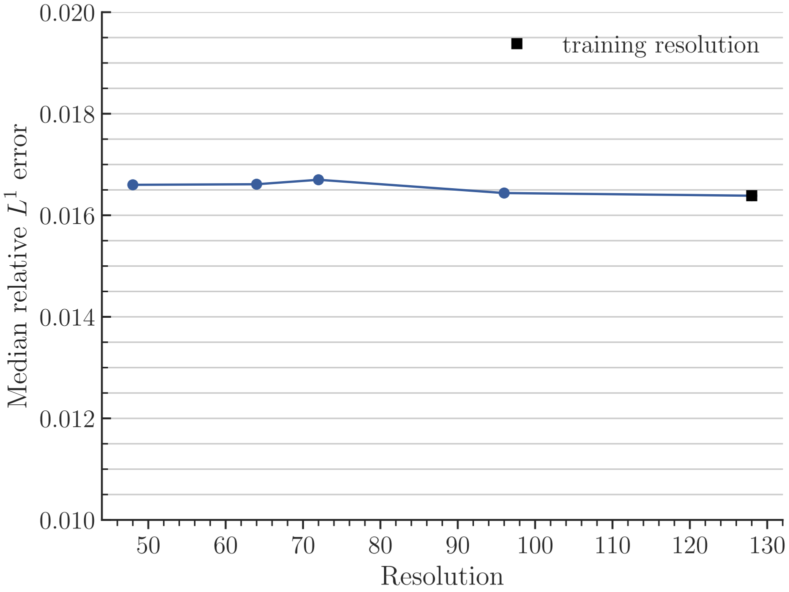


Figure 52: Test performance of POSEIDON-B finetuned on 128 trajectories of the NS-PwC dataset for multiple resolutions.

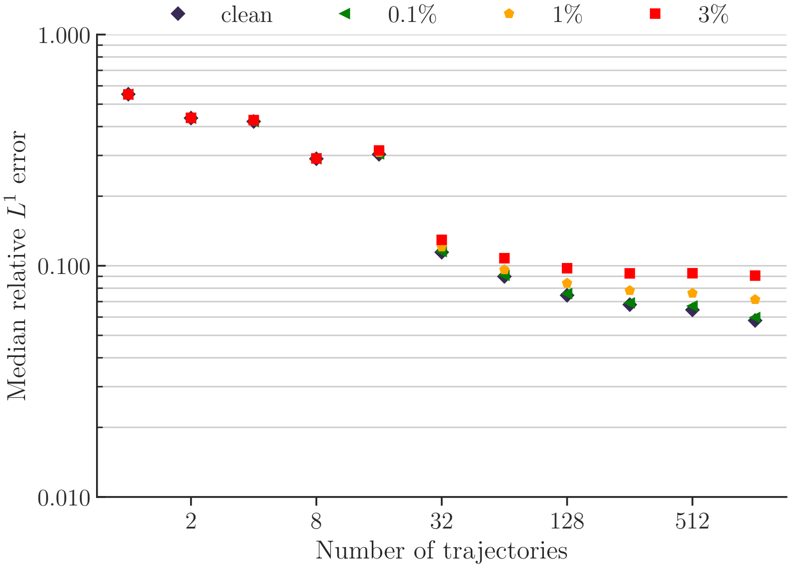


Figure 53: Effect of injecting Gaussian noise in the initial condition on CE-RPUI (before normalizing the data; normalization constants are as before) with POSEIDON-L.

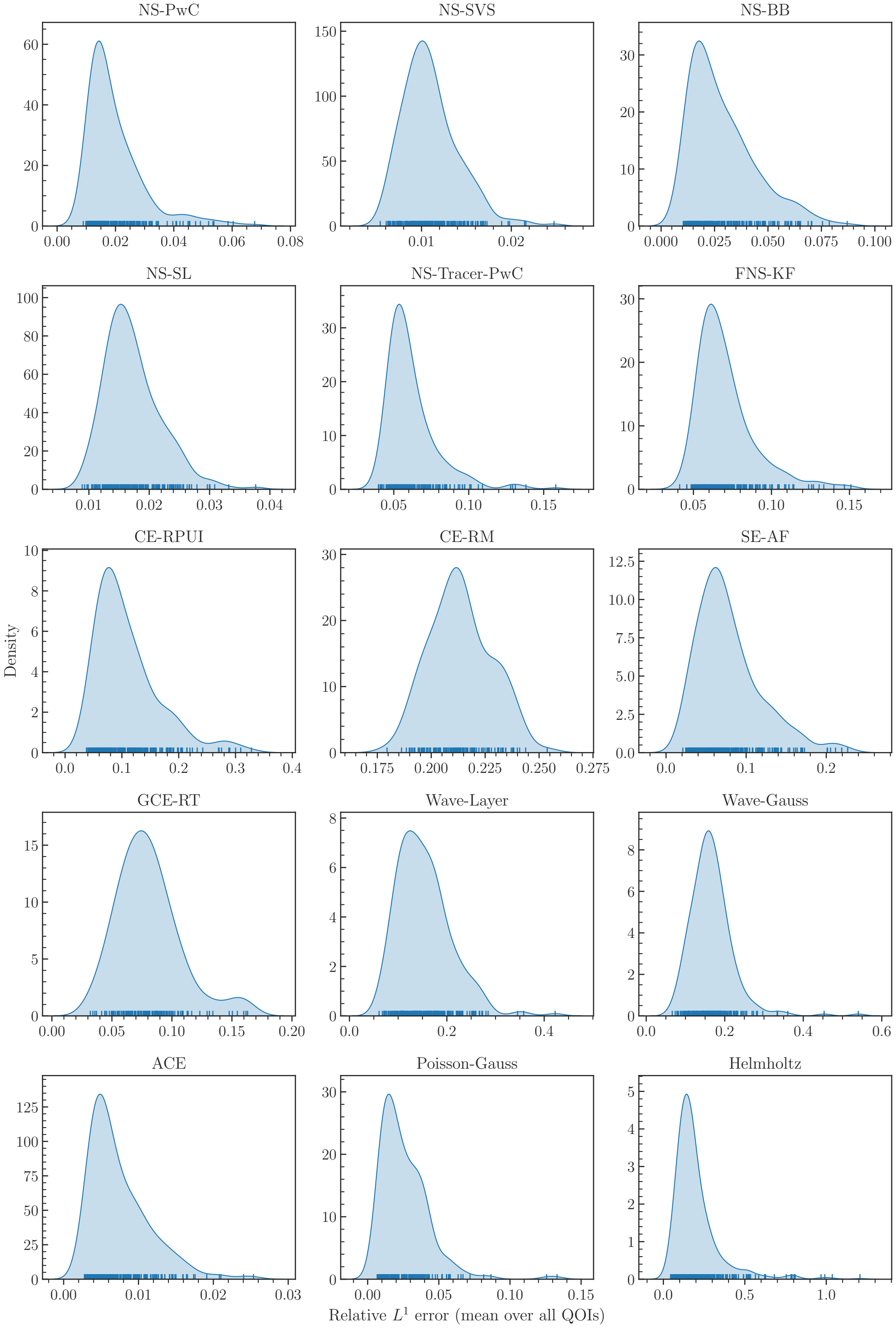


Figure 54: Error distribution of POSEIDON-B finetuned on all downstream tasks (for 128 trajectories in the time-dependent, and 512 in the time-independent case). The kernel density estimate is done over the mean of all functions/quantities of interest.

E Computational Resources

All experiments were run on different types of GPUs, on the Euler cluster of ETH Zurich. Depending on the experiment, we use between 8 and 128 CPU cores and up to 512GB of RAM, with pretrainings using the most CPU cores and RAM. However, we note that this is more than is actually needed, as we tried to minimize being bottlenecked by dataloading. For all our models and baselines, we used consumer-grade GPUs with 24GB of VRAM. All our pretrainings were performed in (data-)parallel on 8 NVIDIA GeForce RTX 4090 GPUs. All finetuning experiments and most scratch trainings were performed on a single GPU, while some scratch training runs with a lot of data were performed in (data-)parallel. Pretraining times can be read off from Table 11.

Table 11: Approximate pretraining times on 8 NVIDIA GeForce RTX 4090 GPUs. Batch sizes are given in parentheses.

POSEIDON-L	POSEIDON-B	POSEIDON-T	CNO-FM
165h (16)	118h (40)	22h (80)	178h (32)

In Table 12, we provide an overview over the inference times of each model for a single call to it. We observe from this table that even the biggest POSEIDON-L has an (average) inference time of less than 10^{-2} secs. This is contrast to the PDE solvers that were used to generate the data in this paper. Their run times, for a resolution of 128^2 ranged from anywhere between 0.1 sec (for highly optimized GPU solver [62] for the NS datasets to 10 secs for FENICS [40] FEM solver for the Poisson-Gauss dataset to approx 100 secs for NEWTUN [45] for generating the airfoils datasets to 500 secs for the well-balanced scheme to generate the GCE-RT. Thus, we observe a gain in inference time from anywhere between $1-5$ orders of magnitude.

Table 12: Approximate inference times (per call and normalized to a single sample) for different models, all reported on a NVIDIA GeForce RTX 4090 GPU for the FNS-KF experiment. Batch sizes are given in parentheses. We note that the values given here are just proxies as this was not tested in a controlled environment.

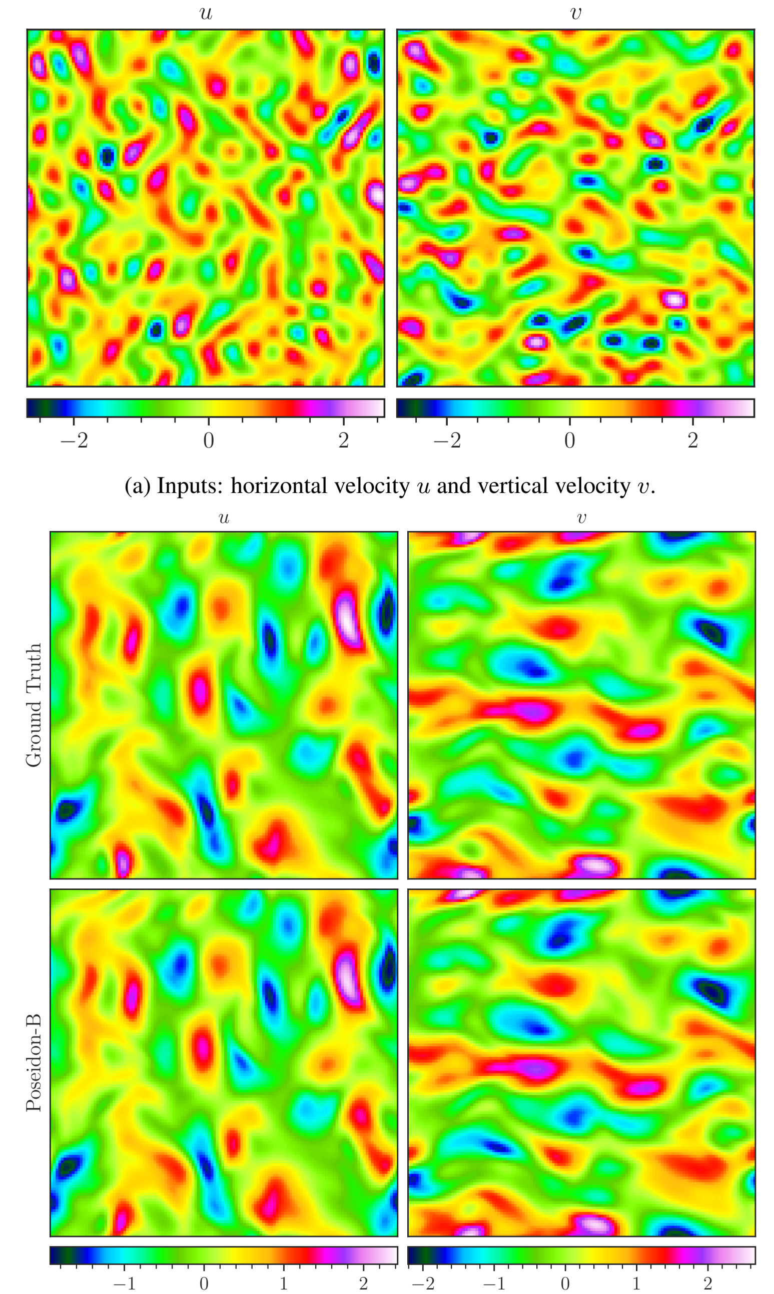
Model	Approximate inference time
POSEIDON-L	4 ms (16)
POSEIDON-B	2.9ms (40)
POSEIDON-T	1.6ms (40)
CNO-FM	1.8ms (32)
MPP-B	10ms (4)
CNO	0.9ms (32)
scOT	3ms (40)
FNO	2ms (40)

F Pretrained Models, Datasets, and Source Code

The source code corresponding to this work is available on Github (https://github.com/camlab-ethz/poseidon). Everything is tightly integrated into Huggingface Transformers [73] and we make heavy use of Huggingface Accelerate for distributed training.

In addition to the code, we make (pretrained) models and datasets available on the Huggingface Hub (https://huggingface.co/camlab-ethz), see the Poseidon collection for pretrained models and pretraining datasets, the Poseidon – Downstream Tasks collection for all downstream tasks, or the PDEGYM collection for all datasets in PDEGYM.

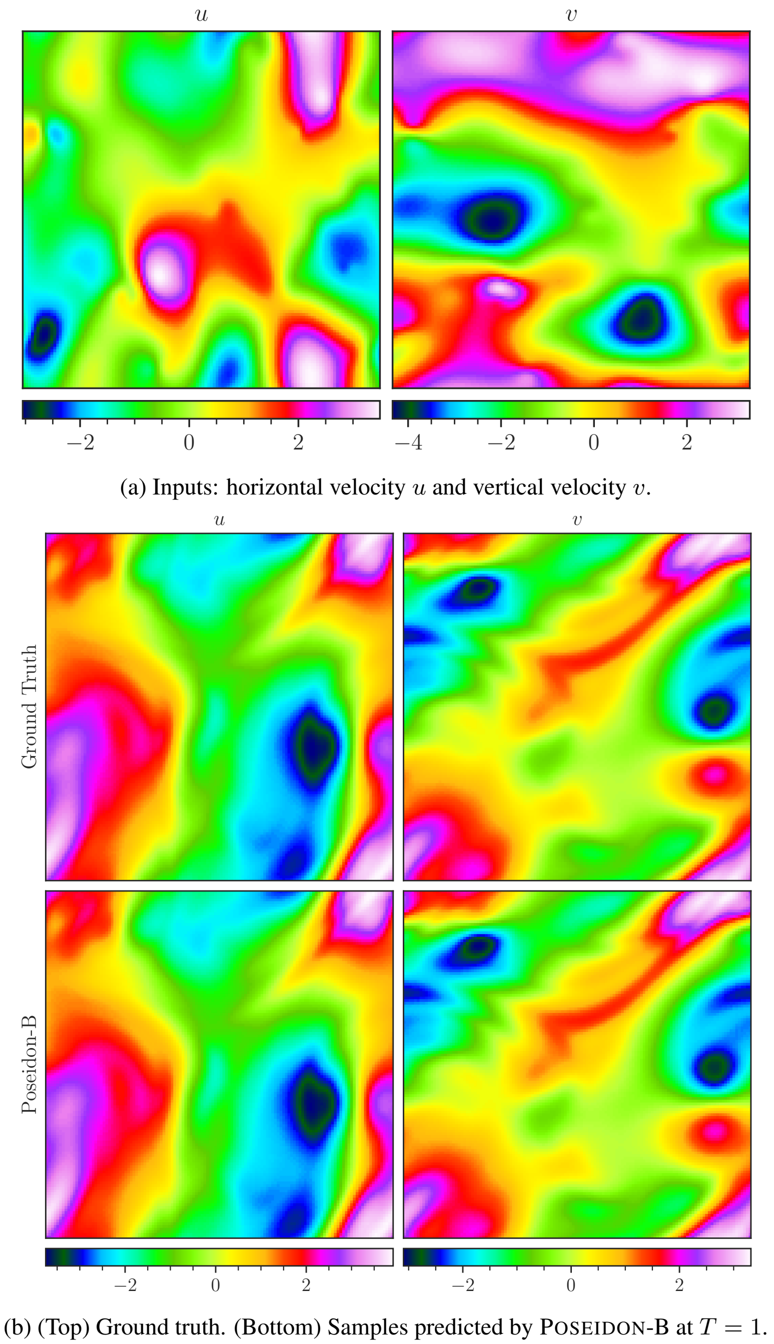
G Visualizations



(a) Inputs: horizontal velocity u and vertical velocity v.

(b) (Top) Ground truth. (Bottom) Samples predicted by POSEIDON-B at $T = 1$.

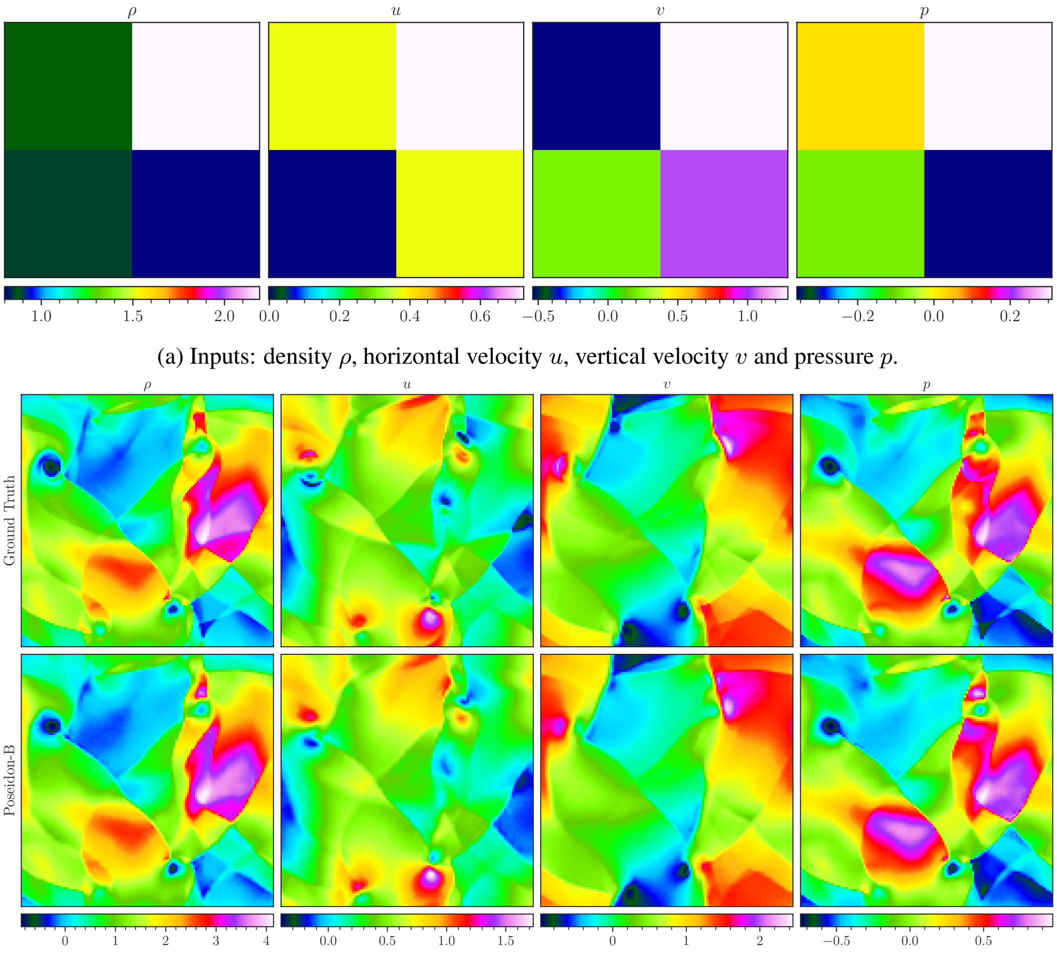
Figure 55: NS-Sines. Visualization of a random sample.



(a) Inputs: horizontal velocity u and vertical velocity v.

(b) (Top) Ground truth. (Bottom) Samples predicted by POSEIDON-B at $T = 1$.

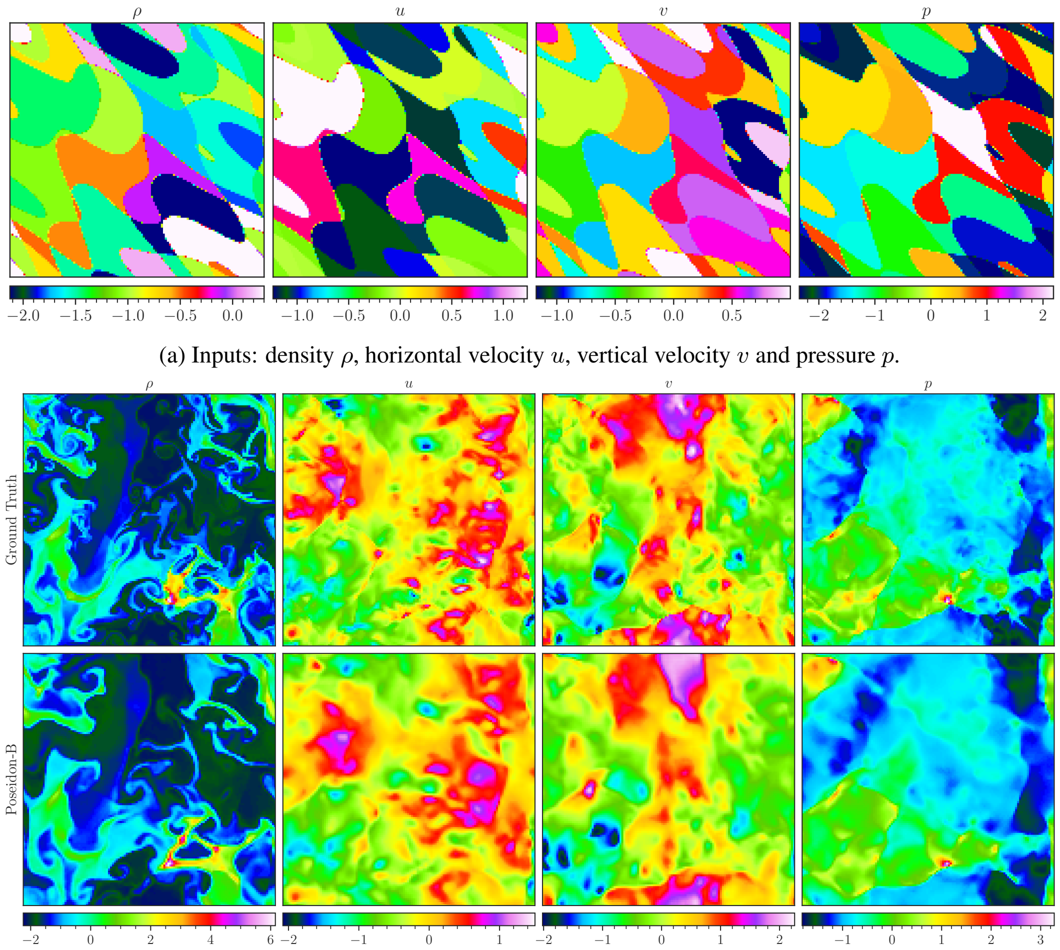
Figure 56: NS-Gauss. Visualization of a random sample.



(a) Inputs: density ρ, horizontal velocity u, vertical velocity v and pressure p.

(b) (Top) Ground truth. (Bottom) Samples predicted by POSEIDON-B at $T = 1$.

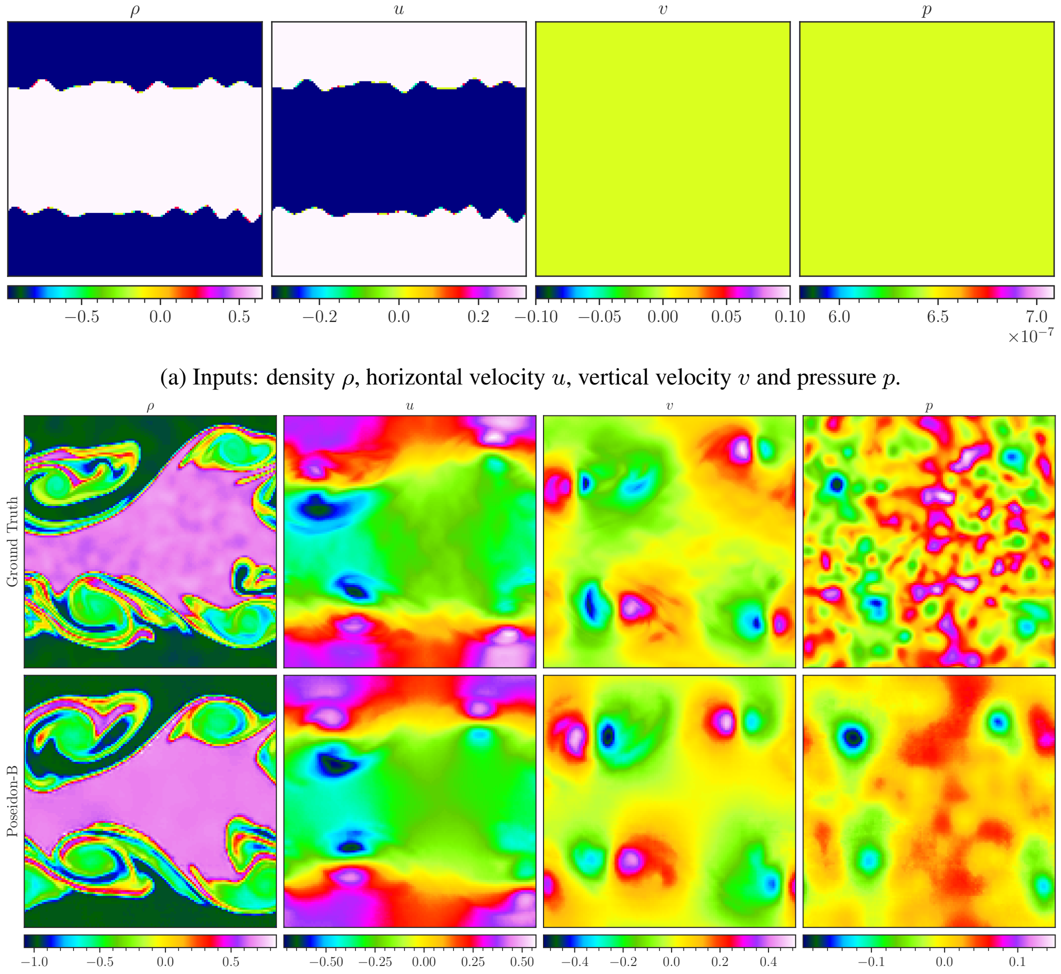
Figure 57: CE-RP. Visualization of a random sample.



(a) Inputs: density ρ, horizontal velocity u, vertical velocity v and pressure p.

(b) (Top) Ground truth. (Bottom) Samples predicted by POSEIDON-B at $T = 1$.

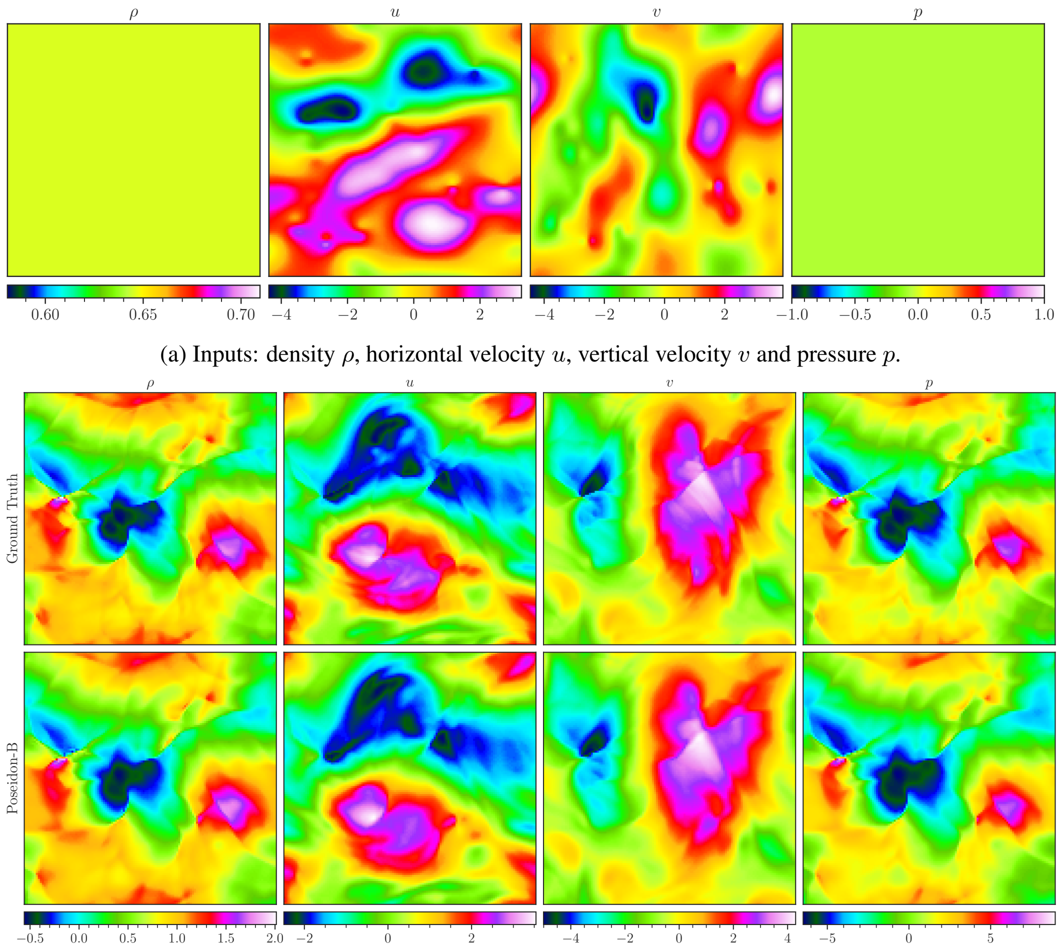
Figure 58: CE-CRP. Visualization of a random sample.



(a) Inputs: density ρ, horizontal velocity u, vertical velocity v and pressure p.

(b) (Top) Ground truth. (Bottom) Samples predicted by POSEIDON-B at $T = 1$.

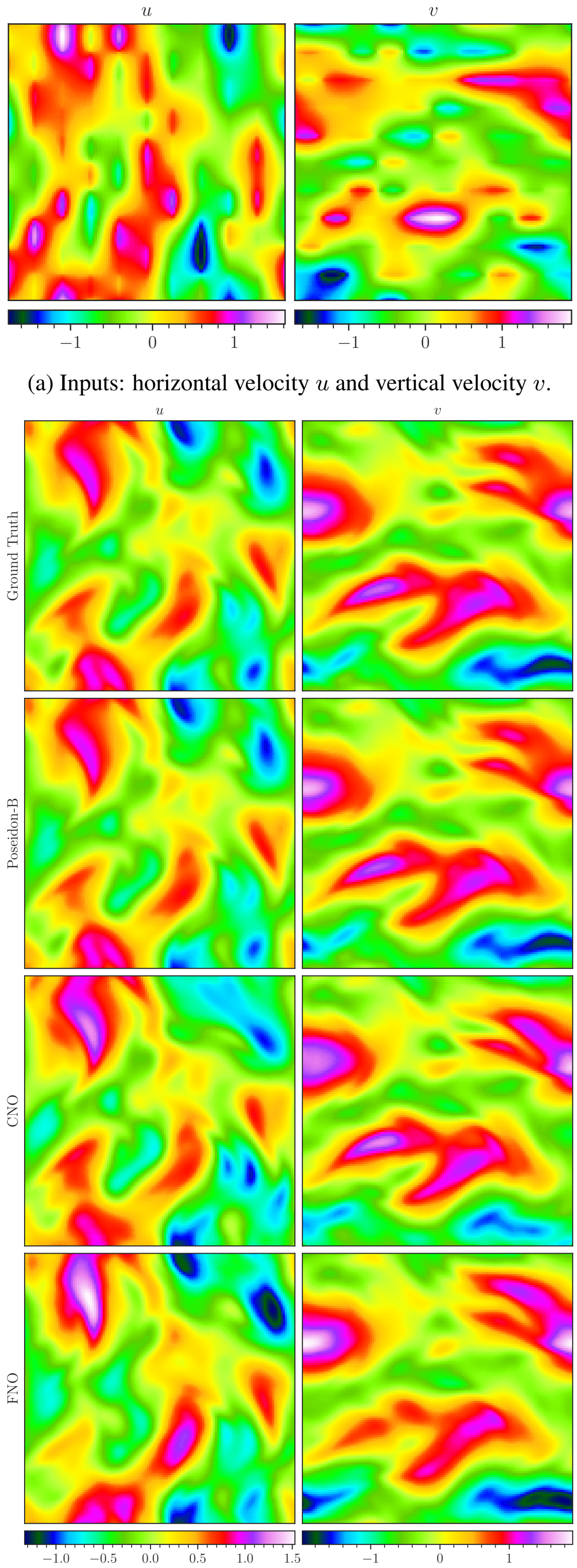
Figure 59: CE-KH. Visualization of a random sample.



(a) Inputs: density ρ, horizontal velocity u, vertical velocity v and pressure p.

(b) (Top) Ground truth. (Bottom) Samples predicted by POSEIDON-B at $T = 1$.

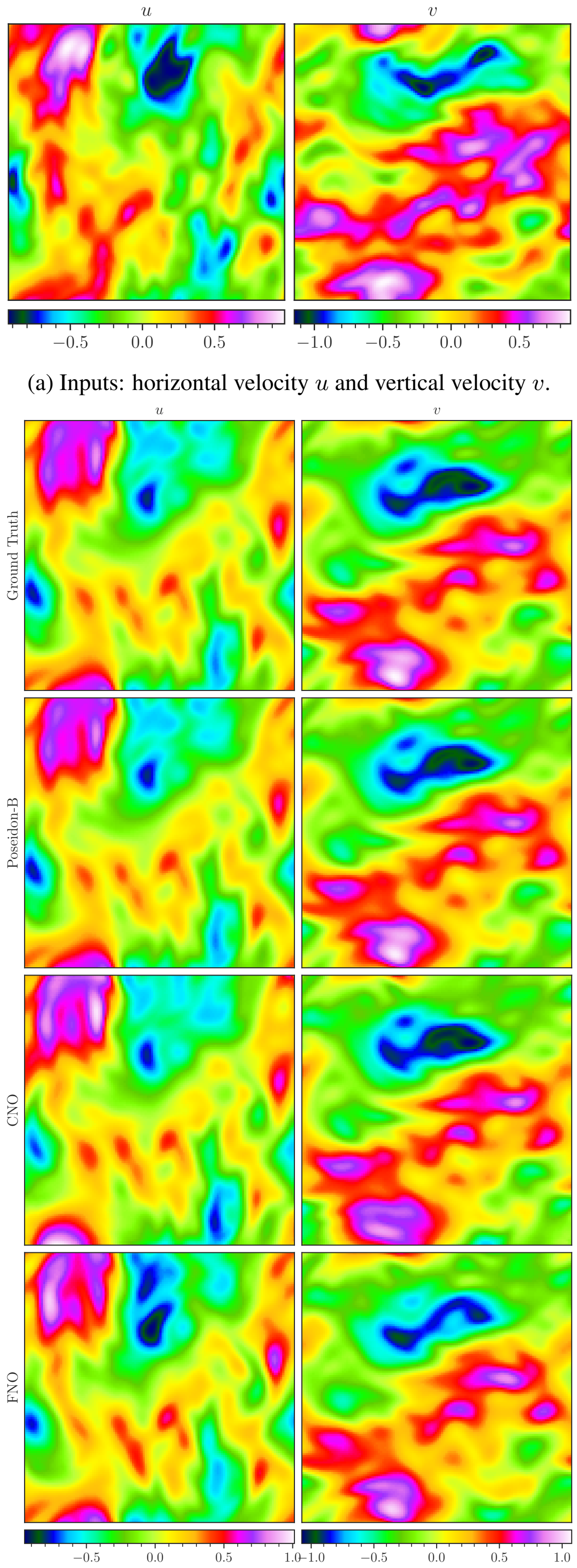
Figure 60: CE-Gauss. Visualization of a random sample.



(a) Inputs: horizontal velocity u and vertical velocity v.

(b) (Top) Ground truth. (From second row onwards) Samples predicted by the finetuned POSEIDON-B, CNO, and FNO at $T = 0.7$.

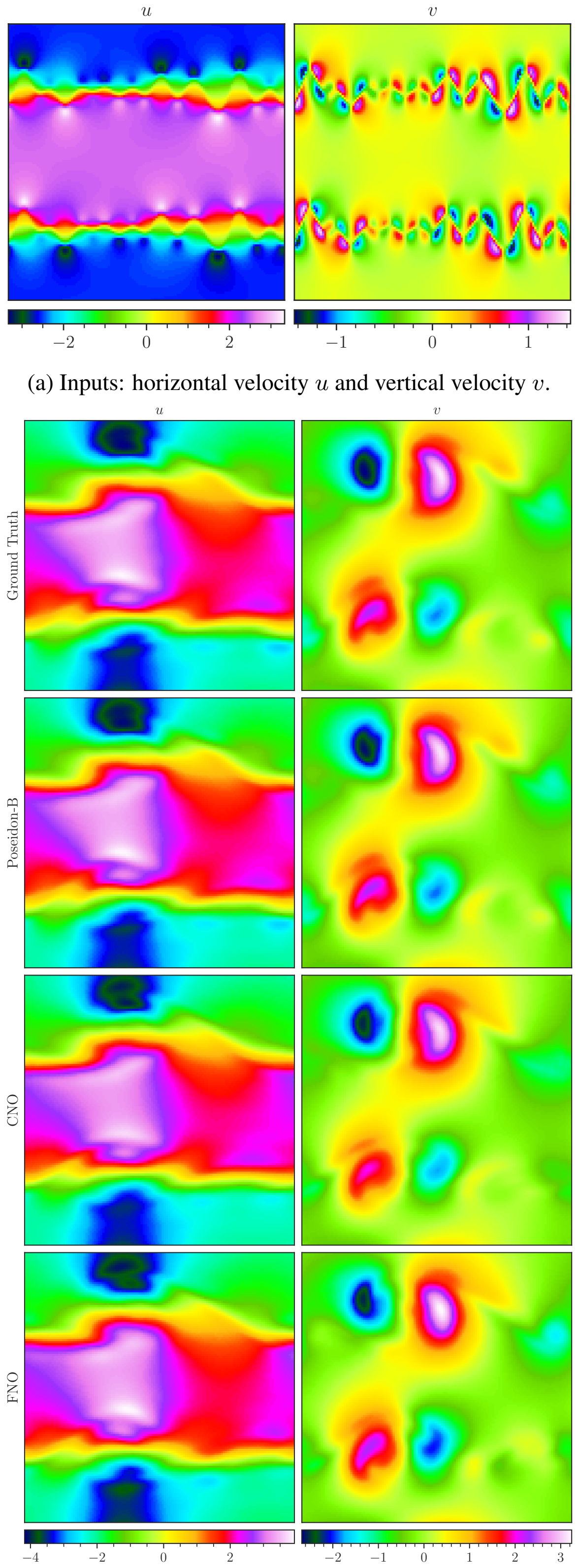
Figure 61: NS-PwC. Visualization of a random sample.



(a) Inputs: horizontal velocity u and vertical velocity v.

(b) (Top) Ground truth. (From second row onwards) Samples predicted by the finetuned POSEIDON-B, CNO, and FNO at $T = 0.7$.

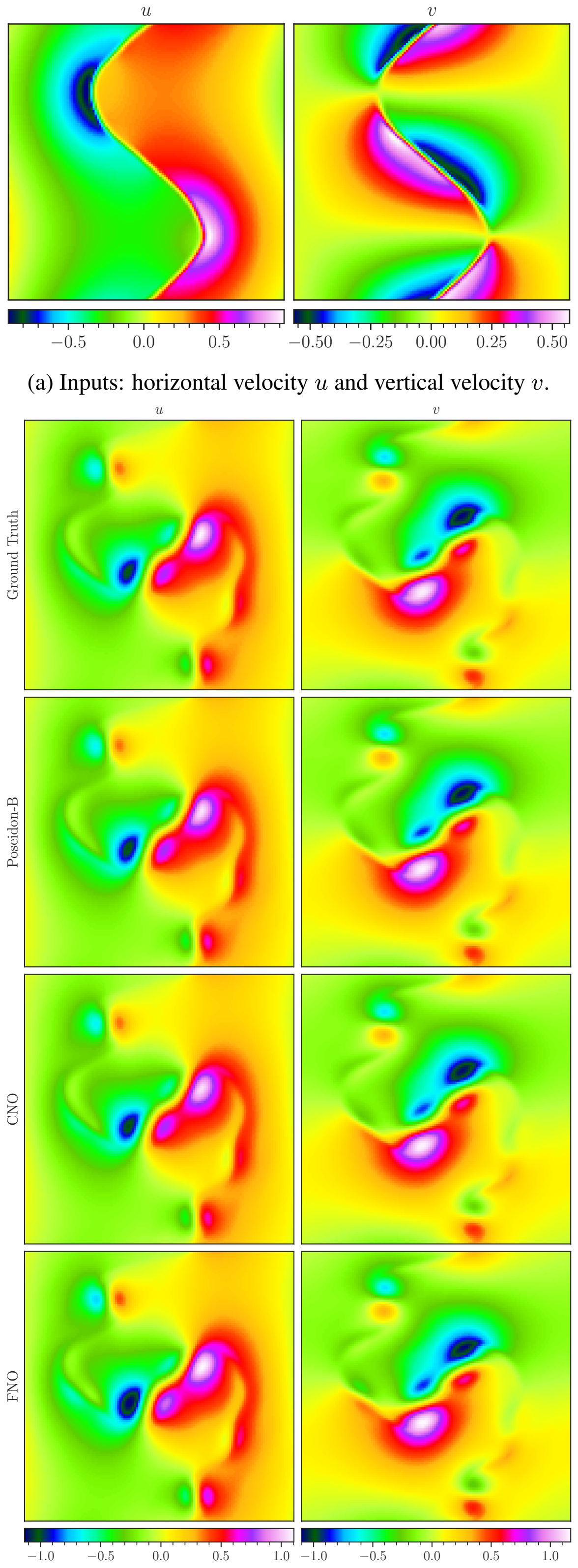
Figure 62: NS-BB. Visualization of a random sample.



(a) Inputs: horizontal velocity u and vertical velocity v.

(b) (Top) Ground truth. (From second row onwards) Samples predicted by the finetuned POSEIDON-B, CNO, and FNO at $T = 0.7$.

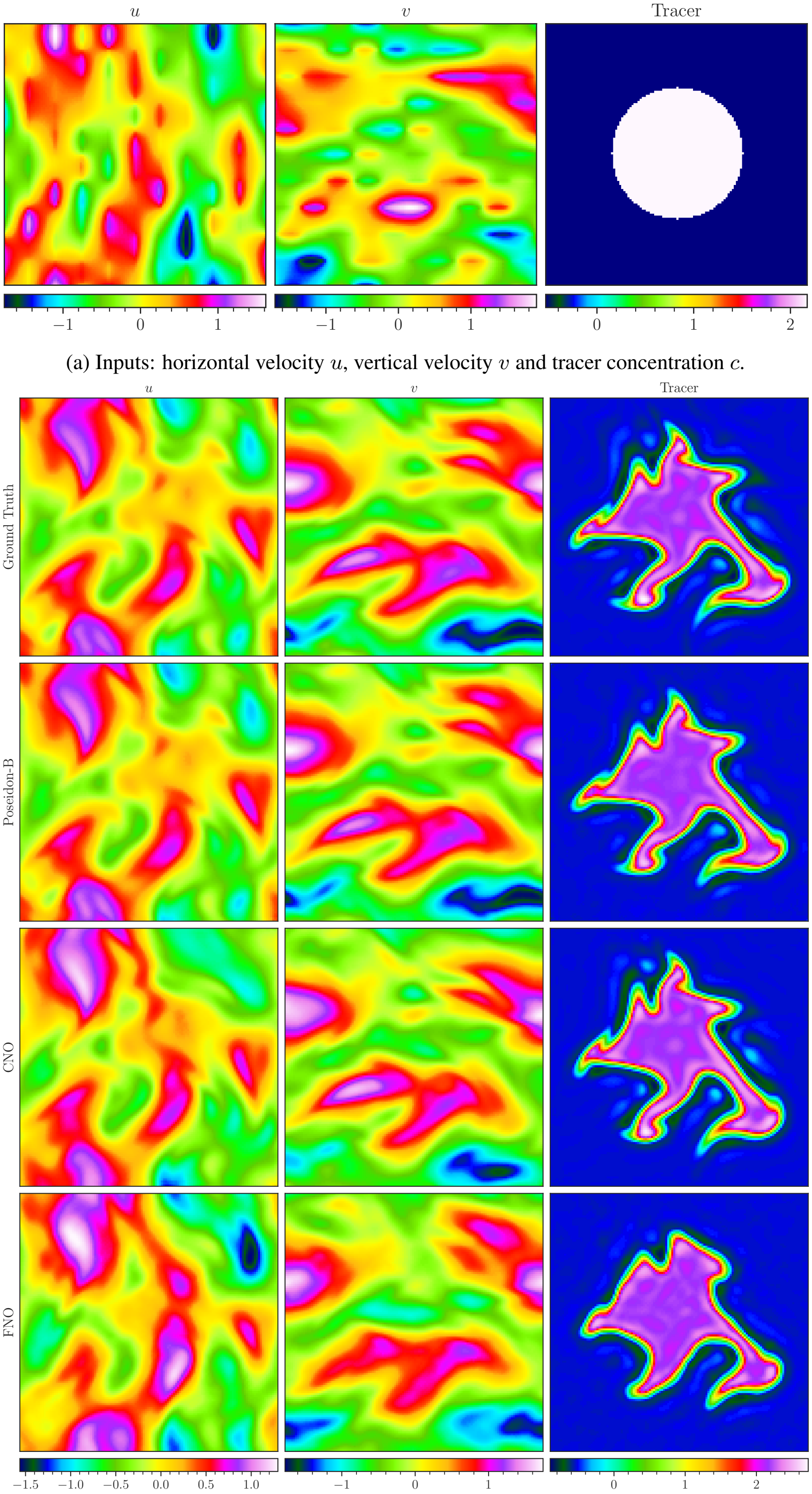
Figure 63: NS-SL. Visualization of a random sample.



(a) Inputs: horizontal velocity u and vertical velocity v.

(b) (Top) Ground truth. (From second row onwards) Samples predicted by the finetuned POSEIDON-B, CNO, and FNO at $T = 0.7$.

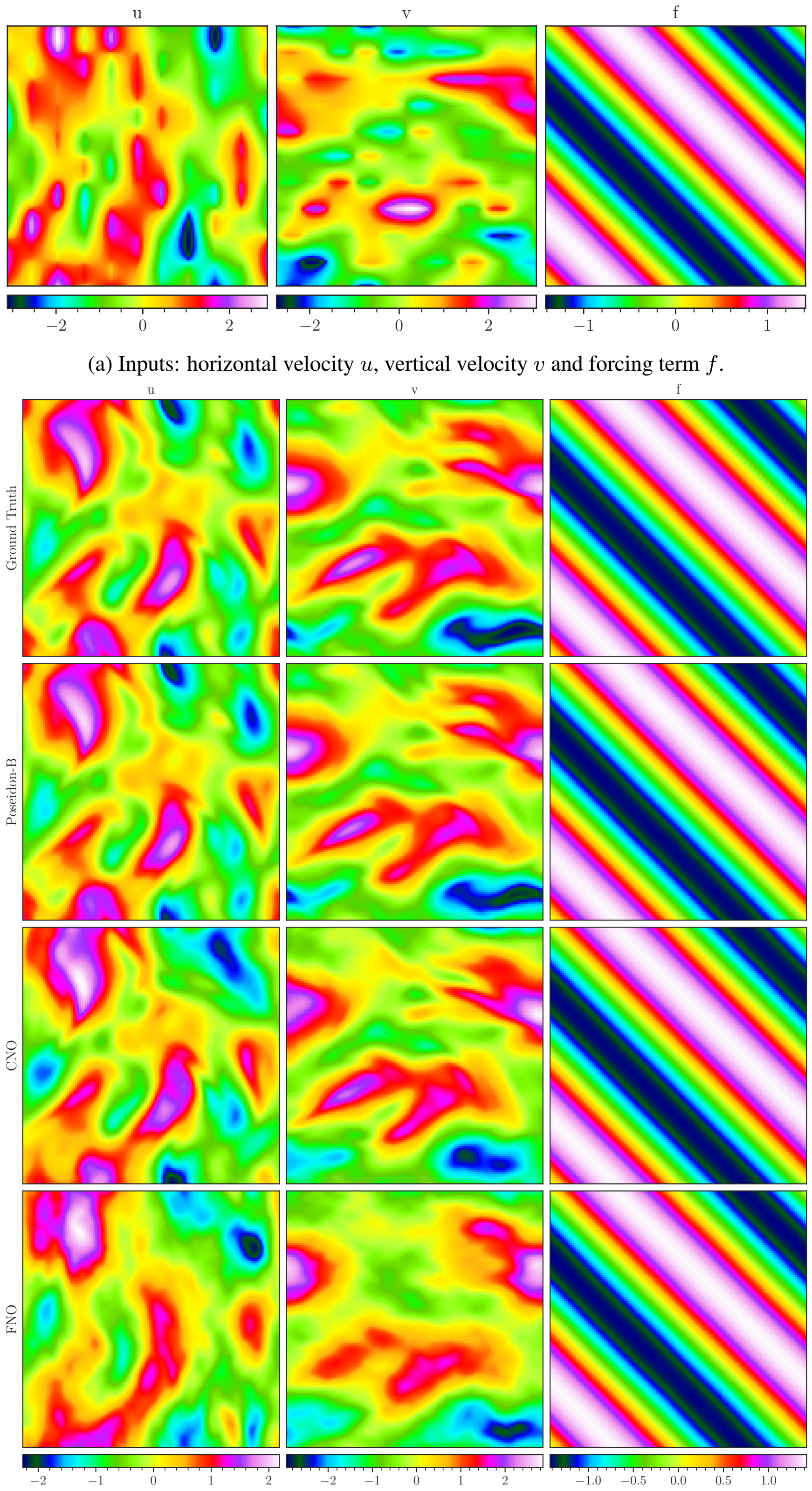
Figure 64: NS-SVS. Visualization of a random sample.



(a) Inputs: horizontal velocity u, vertical velocity v and tracer concentration c.

(b) (Top) Ground truth. (From second row onwards) Samples predicted by the finetuned POSEIDON-B, CNO, and FNO at $T = 0.7$.

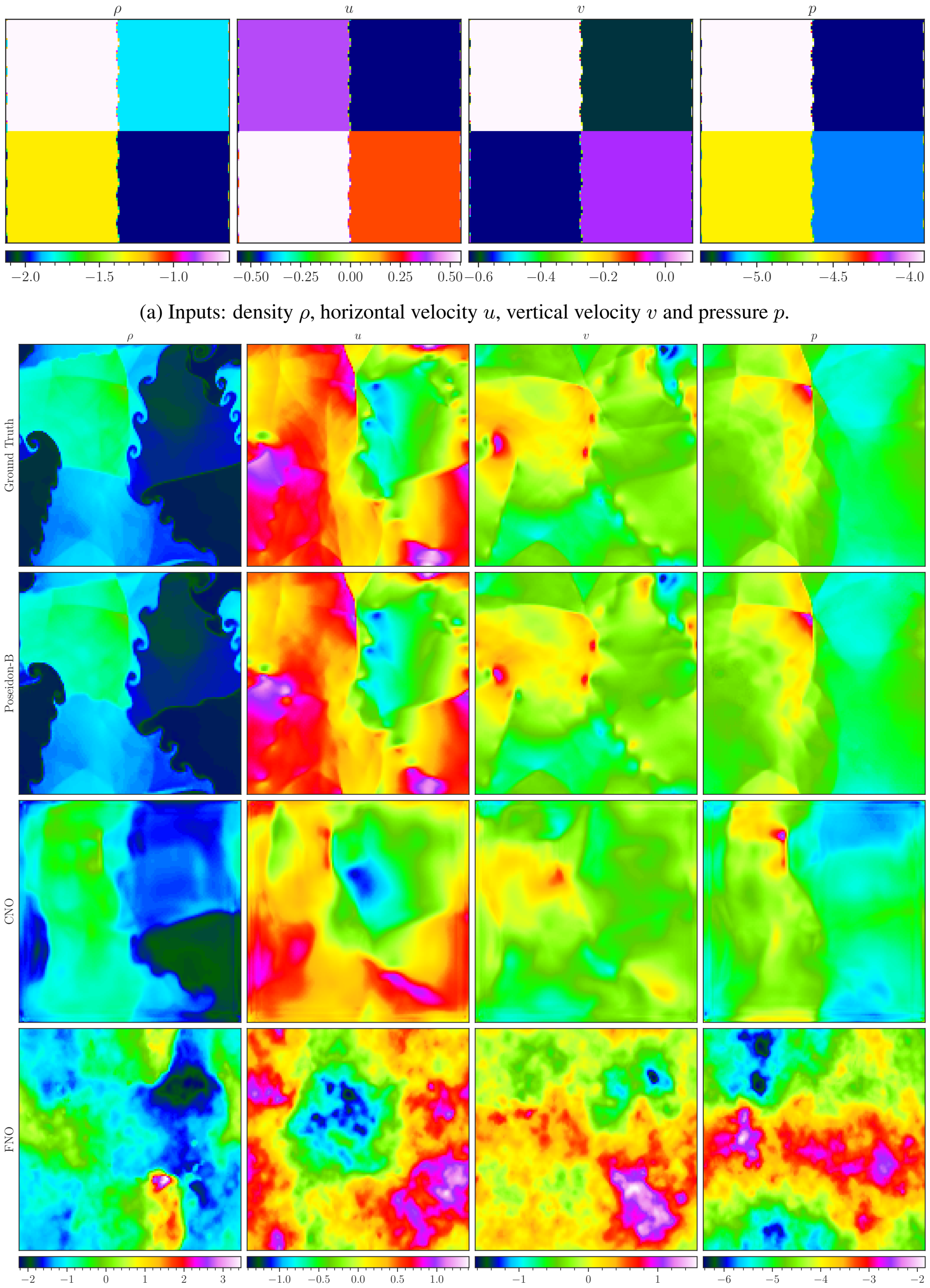
Figure 65: NS-Tracer-PwC. Visualization of a random sample.



(a) Inputs: horizontal velocity u, vertical velocity v and forcing term f.

(b) (Top) Ground truth. (From second row onwards) Samples predicted by the finetuned POSEIDON-B, CNO, and FNO at $T = 0.7$.

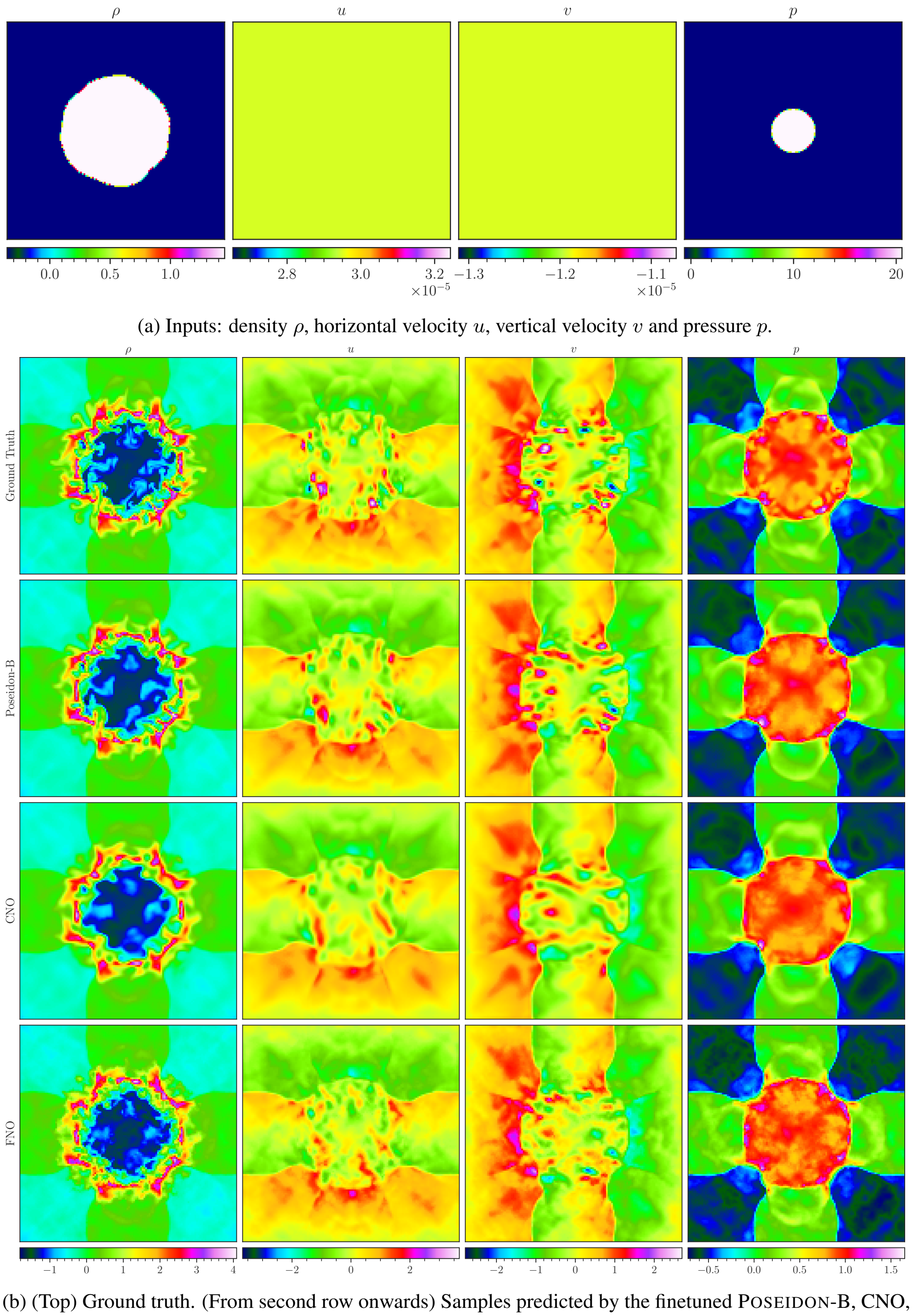
Figure 66: FNS-KF. Visualization of a random sample.



(a) Inputs: density ρ, horizontal velocity u, vertical velocity v and pressure p.

(b) (Top) Ground truth. (From second row onwards) Samples predicted by the finetuned POSEIDON-B, CNO, and FNO at $T = 0.7$.

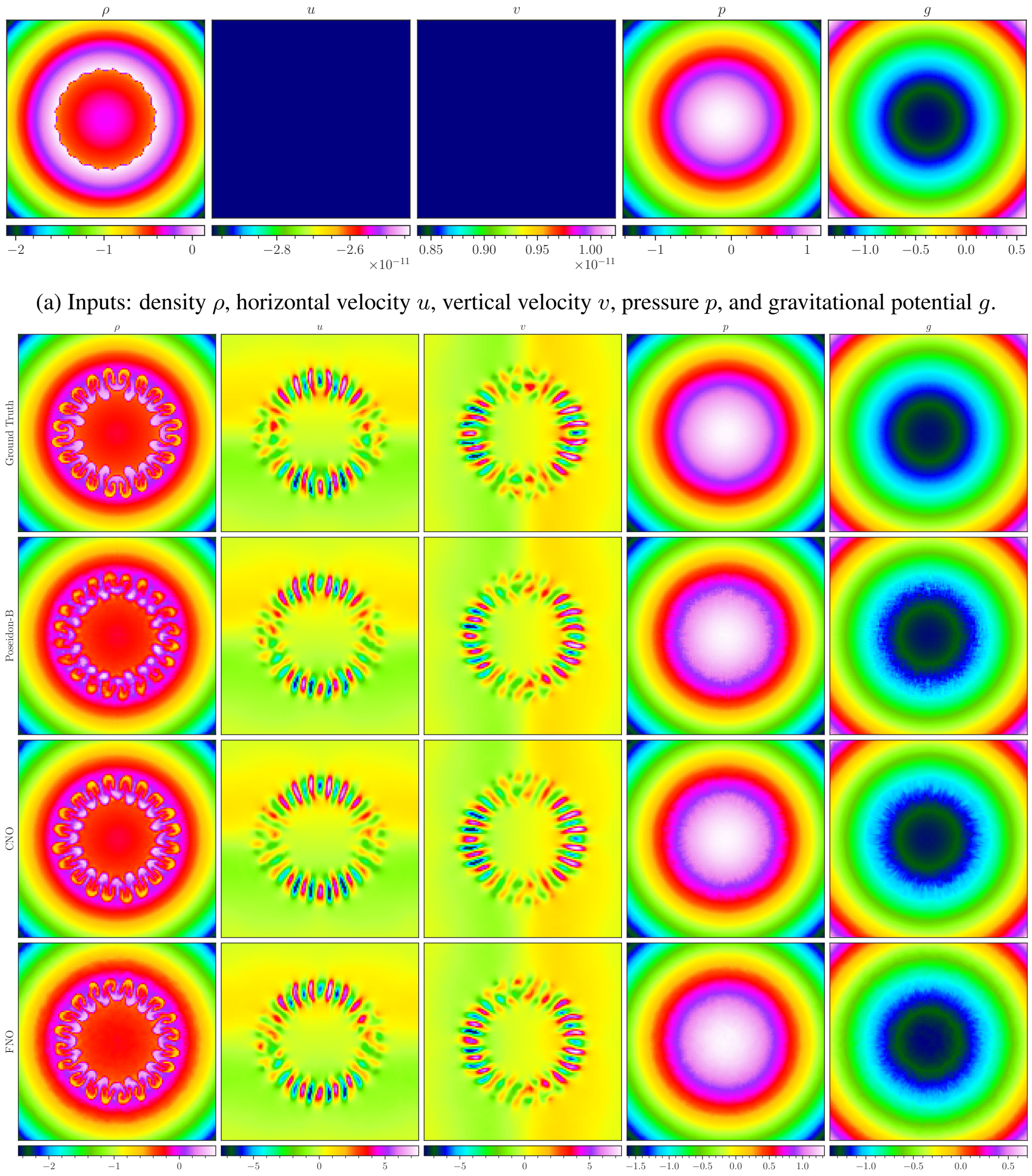
Figure 67: CE-RPUI. Visualization of a random sample.



(a) Inputs: density ρ, horizontal velocity u, vertical velocity v and pressure p.

(b) (Top) Ground truth. (From second row onwards) Samples predicted by the finetuned POSEIDON-B, CNO, and FNO at $T = 1.4$.

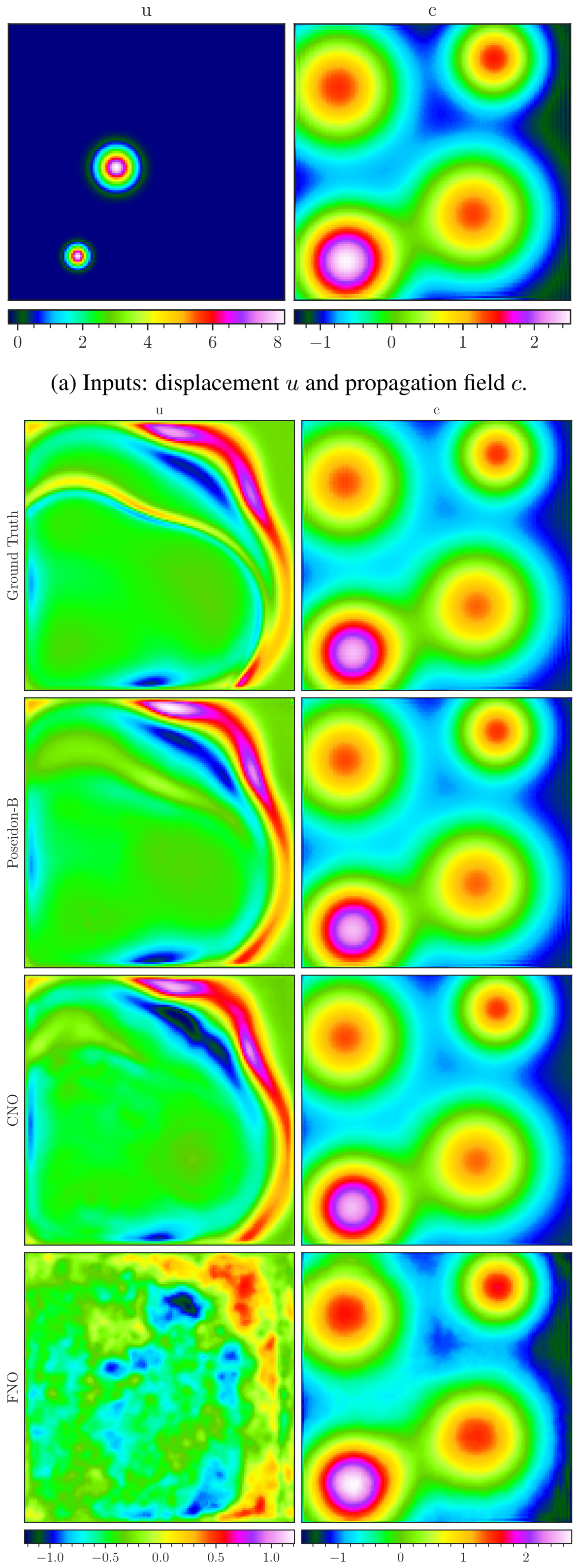
Figure 68: CE-RM. Visualization of a random sample.



(a) Inputs: density ρ, horizontal velocity u, vertical velocity v, pressure p, and gravitational potential g.

(b) (Top) Ground truth. (From second row onwards) Samples predicted by the finetuned POSEIDON-B, CNO, and FNO at the seventh time step.

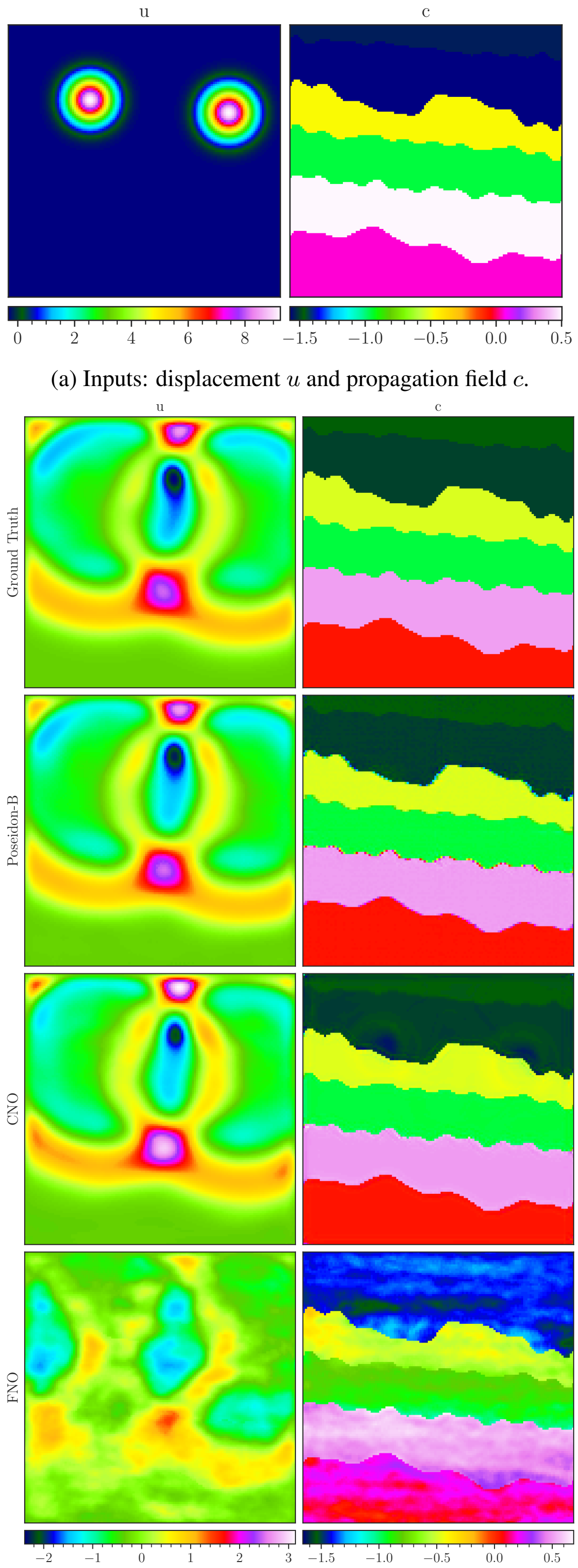
Figure 69: CE-RM. Visualization of a random sample.



(a) Inputs: displacement u and propagation field c.

(b) (Top) Ground truth. (From second row onwards) Samples predicted by the finetuned POSEIDON-B, CNO, and FNO at the 14-th time step.

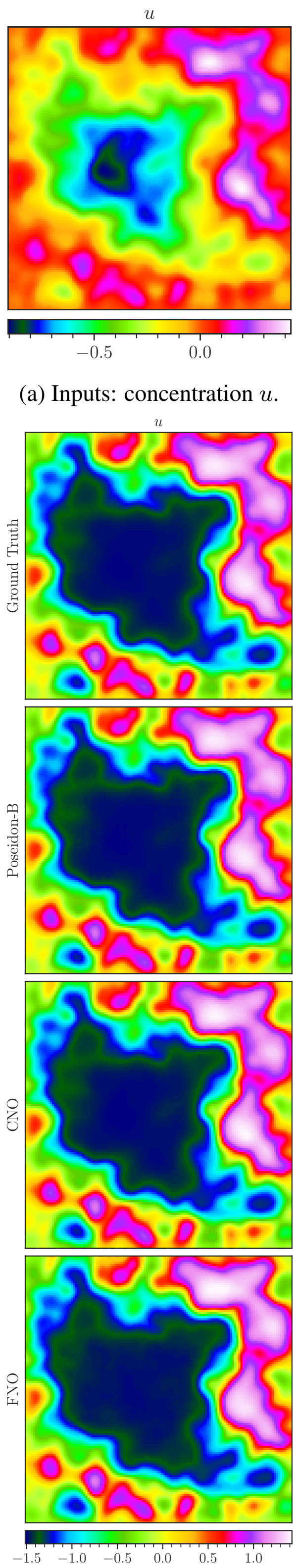
Figure 70: Wave-Gauss. Visualization of a random sample.



(a) Inputs: displacement u and propagation field c.

(b) (Top) Ground truth. (From second row onwards) Samples predicted by the finetuned POSEIDON-B, CNO, and FNO at $T = 0.7$.

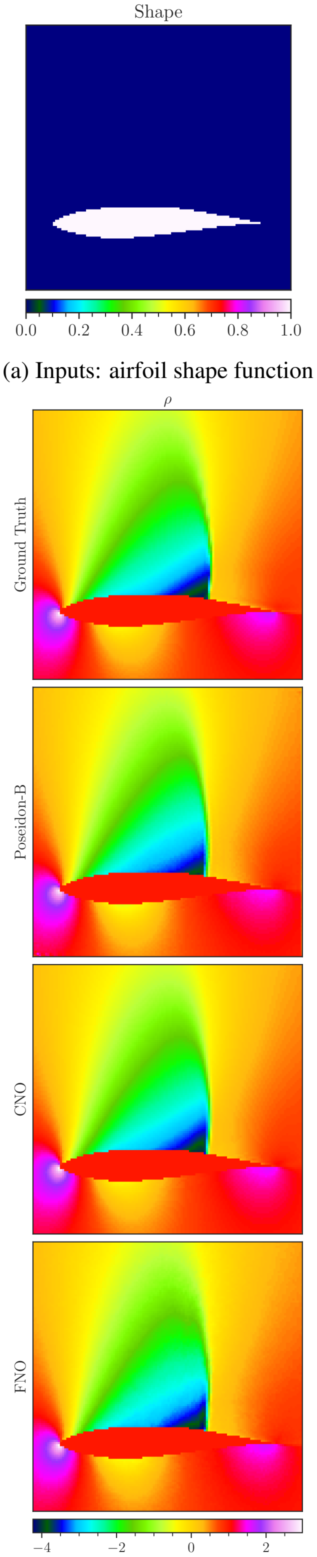
Figure 71: Wave-Layer. Visualization of a random sample.



(a) Inputs: concentration u.

(b) (Top) Ground truth. (From second row onwards) Samples predicted by the finetuned POSEIDON-B, CNO, and FNO at the 14-th time step.

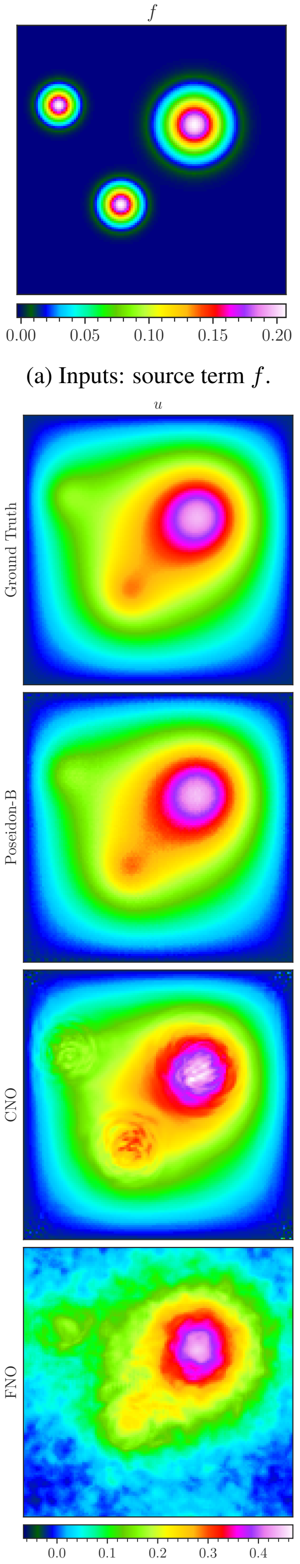
Figure 72: ACE. Visualization of a random sample.



(a) Inputs: airfoil shape function

(b) (Top) Ground truth. (From second row onwards) Samples (density ρ) predicted by the finetuned POSEIDON-B, CNO, and FNO.

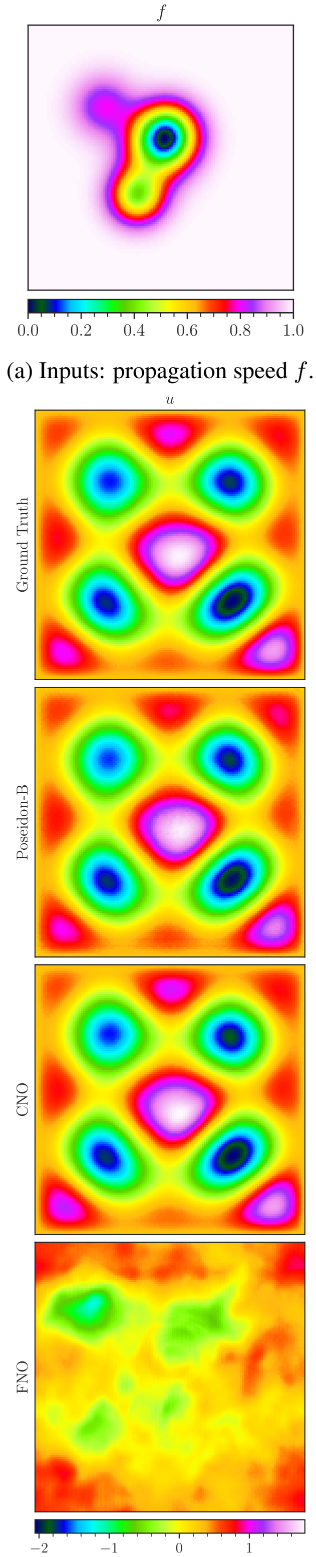
Figure 73: SE-AF. Visualization of a random sample.



(a) Inputs: source term f.

(b) (Top) Ground truth. (From second row onwards) Samples (solution u) predicted by the finetuned POSEIDON-B, CNO, and FNO.

Figure 74: Poisson-Gauss. Visualization of a random sample.



(a) Inputs: propagation speed f.

(b) (Top) Ground truth. (From second row onwards) Samples (solution u) predicted by the finetuned POSEIDON-B, CNO, and FNO.

Figure 75: Helmholtz. Visualization of a random sample.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: In the paper and SM, we thoroughly justified all our claims and contributions. We detailed all models, datasets and training, as well as testing strategies used. Additionally, we supported our assertions with numerous experiments conducted throughout the study.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We dedicated an entire section to discussing the limitations and clearly outlined the next steps to address them.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: The purpose of this paper is to introduce a foundation model for learning the solution operators of PDEs, featuring novel benchmarks, training techniques, and a new paradigm for foundation models for PDEs. Theoretical results will be addressed in future work.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: We will make the weights of our models, along with the codes and benchmarks, open source. We have clearly explained how the models were trained and finetuned, as well as which datasets were used. We also explained clearly how the datasets were generated. By following our instructions, end users will find it fairly easy to reproduce our results.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility.

In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: As explained in the previous answer, we will make the weights of our models, along with the codes and benchmarks, open source. We explained clearly how the datasets were generated. We will not release the datasets and pretrained checkpoints during the review process as the files are very large and we are not aware of a file hosting service that enables anonymous release of this size. Code is available at the reviewer's disposal.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We dedicated many sections in the main paper and the SM to explain the training and test details, along with all the relevant information necessary to understand our results.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

Justification: In Section D.6.8, we presented histograms of errors for various tasks. All the relevant statistical information can be inferred from these histograms. All other plots would get too cluttered by adding error bars.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provided an estimation about the training times and resources that we used in all the experiments.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Our research respects the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [NA]

Justification: Our paper is exclusively dedicated to advancing academic research in the area of Partial Differential Equations. Our models are tailored for utilization by researchers with an interest in this domain.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: To the best of our knowledge, our paper does not present any identifiable safety risks.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The segments of code not authored by us, such as the CNO filtering, are explicitly acknowledged within the codebase, and due credit is accorded to the respective owners of these assets.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: All the new assets are well documented. Detailed explanations of the datasets and codes are provided.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: This paper involves neither crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: This paper does not involve research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.