HyPINO: Multi-Physics Neural Operators via HyperPINNs and the Method of Manufactured Solutions

Rafael Bischof[1] **Michal Piovarči**[1] **Michael A. Kraus**[2]

Siddhartha Mishra[3] **Bernd Bickel**[1]

[1]Computational Design Lab, ETH Zurich, Switzerland
[2]Institute of Structural Mechanics and Design, TU Darmstadt, Germany
[3]Seminar for Applied Mathematics, ETH Zurich, Switzerland
*Correspondence to rabischof@ethz.ch

Abstract

We present HyPINO, a multi-physics neural operator designed for zero-shot generalization across a broad class of PDEs without requiring task-specific fine-tuning. Our approach combines a Swin Transformer-based hypernetwork with mixed supervision: (i) labeled data from analytical solutions generated via the Method of Manufactured Solutions (MMS), and (ii) unlabeled samples optimized using physics-informed objectives. The model maps PDE parameterizations to target Physics-Informed Neural Networks (PINNs) and can handle linear elliptic, hyperbolic, and parabolic equations in two dimensions with varying source terms, geometries, and mixed Dirichlet/Neumann boundary conditions, including interior boundaries. HyPINO achieves strong zero-shot accuracy on seven benchmark problems from PINN literature, outperforming U-Nets, Poseidon, and Physics-Informed Neural Operators (PINO). Further, we introduce an iterative refinement procedure that treats the residual of the generated PINN as "delta PDE" and performs another forward pass to generate a corrective PINN. Summing their contributions and repeating this process forms an ensemble whose combined solution progressively reduces the error on six benchmarks and achieves a >100× lower L_2 loss in the best case, while retaining forward-only inference. Additionally, we evaluate the fine-tuning behavior of PINNs initialized by HyPINO and show that they converge faster and to lower final error than both randomly initialized and Reptile-meta-learned PINNs on five benchmarks, performing on par on the remaining two. Our results highlight the potential of this scalable approach as a foundation for extending neural operators toward solving increasingly complex, nonlinear, and high-dimensional PDE problems. The code and model weights are publicly available at `https://github.com/rbischof/hypino`.

1 Introduction

Neural operators have emerged as a promising paradigm for solving partial differential equations (PDEs). Their ability to generalize across families of PDEs, fast inference, and full differentiability make them appealing for a wide range of scientific computing tasks. In the longer term, such methods may serve as building blocks for general-purpose, foundational, and multi-physics simulators, sometimes referred to as "world-model predictors" [9, 26, 50, 53].

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

However, existing neural operators are typically sample inefficient [19]. As a result, most prior work focuses on narrowly defined problem families [31]. Variations are limited to singular aspects such as specific PDE parameters (e.g., diffusion coefficients) [6], boundary conditions [10], or domain shapes [55]. The support for simultaneous variations of PDE operators remains limited to subdomains, such as parametrized convection-diffusion-reaction PDEs [52].

One way to address the data requirement is by incorporating physics-informed losses. Such losses can provide self-supervision without requiring labeled simulation data [40, 47]. While promising, existing methods often suffer from spectral bias [48] and mode collapse [51]. Moreover, purely physics-based training is unstable in practice [12]. Therefore, even with physics-based losses, obtaining a large, labeled dataset that spans a diverse range of PDEs remains a significant bottleneck.

To overcome these challenges, we propose HyPINO, a hybrid framework that combines physics-informed learning with synthetic supervised data. Our approach leverages a Swin Transformer-based hypernetwork [8, 13, 30] to map PDE specifications to the parameters of a target physics-informed neural network (PINN). HyPINO enables zero-shot generalization without task-specific fine-tuning. A key contribution of our work is a scalable synthetic data pipeline that generates two complementary types of training data: (i) Supervised samples generated via the Method of Manufactured Solutions (MMS) [38] by selecting target solutions and deriving the corresponding PDEs analytically. These provide direct supervision with known reference solutions. (ii) Physics-only samples constructed by randomly sampling PDE operators, source terms, and boundary / initial conditions. These are trained using physics-informed losses without requiring ground-truth solutions. This hybrid training strategy allows us to cover a broad spectrum of two-dimensional linear elliptic, hyperbolic, and parabolic PDEs with mixed Dirichlet and Neumann boundary conditions on complex domain geometries, spanning a wide range of phenomena in the natural and engineering sciences, including heat diffusion, wave propagation, acoustic scattering, and membrane deformation.

In addition, we introduce an iterative refinement procedure that builds an ensemble of corrective PINNs. At each iteration, the model evaluates the residual error and generates a "delta" PINN to improve the solution. This ensemble refinement provides a lightweight alternative to traditional fine-tuning, requiring only inference passes rather than full backward passes.

We evaluate HyPINO on seven diverse PDE benchmarks, demonstrating improved zero-shot generalization compared to baselines such as U-Nets [41], Poseidon [19], and PINO [29]. We also find that PINNs initialized with our method fine-tune more efficiently than those starting from random or meta-learned initializations, achieving faster convergence and lower final errors.

In summary, our main contributions are (i) a hybrid physics-informed and supervised learning framework for multi-physics PDE solving, (ii) a scalable data generation pipeline combining random physics sampling with MMS-based supervised examples, (iii) an ensemble-based refinement mechanism that improves prediction quality without expensive retraining, and (iv) empirical results showing strong zero-shot and fine-tuning performance across multiple PDE benchmarks compared to SOTA.

We believe these contributions offer a practical step toward more general-purpose, data-efficient neural operators for multi-physics problems and world simulator foundation models.

2 Related Work

Neural operators aim to approximate solution operators that map PDE specifications to continuous solution fields, enabling fast, mesh-free inference and generalization to unseen problem instances [15, 27, 28, 29, 31, 34]. Recent work scales these ideas toward foundation models that ingest large corpora of simulated data or equation specifications and promise broad cross-task transfer [17, 19, 35, 43, 52]. Despite rapid progress, most operators still target narrow PDE families (e.g. fixed equations with varying coefficients) and depend on expensive high-fidelity solvers for supervision [52]. Embedding the governing equations in the loss function alleviates the need for labeled data and improves physical fidelity [40, 47]. While the original formulation was introduced for stand-alone Physics-Informed Neural Networks (PINNs), the same residual losses have recently been integrated into operator architectures, yielding Physics-Informed Neural Operators (PINO) that train from unlabeled residual samples [3, 12, 29]. These approaches still require careful weighting of supervision terms and often struggle with stability and spectral bias for complex PDEs.

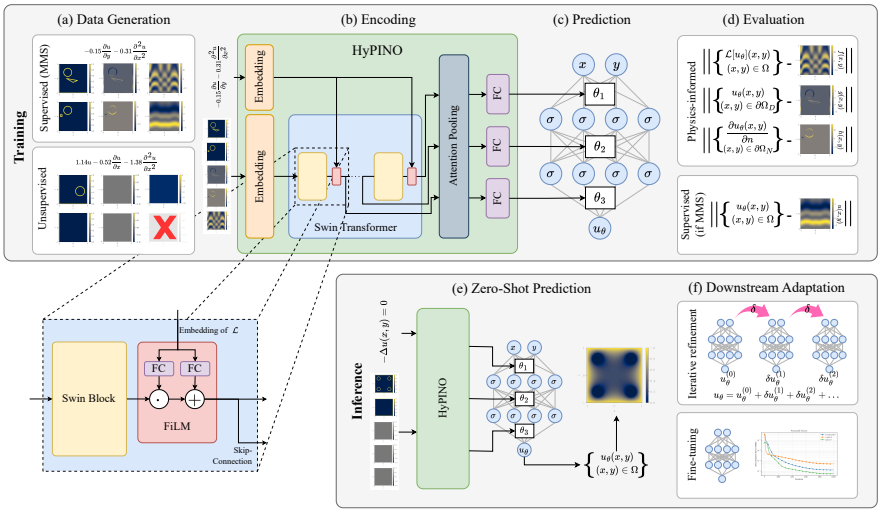


Figure 1: **Overview of the HyPINO pipeline.** (a) Training data includes supervised samples from MMS and unsupervised physics-informed samples without ground truth. (b) PDEs are encoded as multi-channel and vector-based inputs and processed by HyPINO to produce task-specific PINN weights. (c) The predicted PINN maps spatial coordinates to the solution field. (d) Training combines physics-informed residual losses as well as supervised losses for MMS data. (e) At inference, HyPINO enables zero-shot prediction for unseen PDEs. (f) Downstream adaptation includes iterative refinement using residual corrections or optional, task-specific fine-tuning.

Hypernetworks generate the parameters of a target network conditioned on an auxiliary input [13]. In the PDE context, HyperPINNs predict PINN weights for varying coefficients [8, 24], and subsequent works extend this idea to boundary conditions, domain changes, and low-rank weight modulation [5, 10, 14, 36]. Yet, existing models rarely support concurrent variation of multiple operators, geometries, and boundary types without task-specific fine-tuning.

The Method of Manufactured Solutions (MMS) provides analytic ground-truth pairs by choosing a target field and deriving the corresponding source term and boundary data [38]. MMS has long served for numerical-solver verification and was recently adopted for PINN evaluation [23] and operator training [18]. However, prior studies focus on single equations (e.g. Poisson); leveraging MMS for *multi-physics* operator pre-training remains largely unexplored.

Our work situates itself at the intersection of these lines: we couple a Swin Transformer hypernetwork with mixed MMS and physics-informed supervision to produce a single model that generalizes zero-shot across diverse linear, elliptic, hyperbolic, and parabolic PDEs with mixed boundary conditions and diverse 2D geometries.

3 Methodology

We consider a family of second-order linear PDEs defined over a bounded domain $\Omega \subset \mathbb{R}^2$ with boundary $\partial\Omega = \partial\Omega_D \cup \partial\Omega_N$, where $\partial\Omega_D$ and $\partial\Omega_N$ denote the Dirichlet and Neumann boundaries, respectively. The goal is to find a function $u : \Omega \to \mathbb{R}^m$ satisfying

$$\mathcal{L}[u](\mathbf{x}) = f(\mathbf{x}) \quad \text{in } \Omega, \quad u(\mathbf{x}) = g(\mathbf{x}) \quad \text{on } \partial\Omega_D, \quad \frac{\partial u}{\partial n}(\mathbf{x}) = h(\mathbf{x}) \quad \text{on } \partial\Omega_N, \tag{1}$$

where $\mathcal{L}$ is a linear differential operator involving derivatives up to second order, $f : \Omega \to \mathbb{R}^m$ is a known source term, and g, h are prescribed boundary functions. Our objective is to learn the solution operator that maps the tuple $(\mathcal{L}, f, g, h)$ to the solution u.

3.1 PDE Parameterization

To support a wide range of linear PDEs while maintaining compatibility with modern machine learning models, we adopt a parameterization that is flexible, user-friendly, and efficiently processed by state-of-the-art architectures. The function f is discretized on a uniform grid over Ω, resulting in a 2D array F representing its values at grid points. The boundary conditions are parametrized by creating two 2D grids per boundary type ($\partial\Omega_D$, $\partial\Omega_N$): (i) A binary mask M indicating the presence of the boundary at each grid point, where we assign a value of 1 to the four grid cells closest to each boundary point and zero elsewhere; and (ii) a value grid V storing the corresponding boundary values (g for Dirichlet or h for Neumann conditions) at those marked cells, with zeros elsewhere. Figure 2 illustrates a sampled PDE instance with its full parameterization. Finally, following [21], we parameterize $\mathcal{L}$ as $\mathcal{L}[u](\mathbf{x}) = c_1u + c_2u_x + c_3u_y + c_4u_{xx} + c_5u_{yy}$, where $\mathbf{c} = (c_1, c_2, c_3, c_4, c_5) \in \mathbb{R}^5$ encodes the operator coefficients.

3.2 Neural Operator Architecture

We base our model on HyperPINN [8], a hypernetwork-based neural operator that maps a parametrized PDE instance to the weights θ of a PINN u_θ specialized to that instance. Formally, the hypernetwork realizes a mapping

$$\big(\mathbf{c},\ F,\ M_g,\ M_h,\ V_g,\ V_h\big) \longmapsto \theta^\star \quad \text{such that} \quad u_{\theta^\star} \approx u, \tag{2}$$

where $\mathbf{c}$ denotes the vector of PDE coefficients, F the discretized source function, M_g and M_h the Dirichlet and Neumann boundary condition location grids, V_g and V_h the Dirichlet and Neumann boundary condition value grids, and u the reference solution.

The vector of operator coefficients $\mathbf{c} \in \mathbb{R}^5$ is embedded into a fixed-length representation $z_C \in \mathbb{R}^{d_C}$ using a Fourier feature mapping [44] which was shown to prevent spectral bias and mode collapse, in particular in physics-informed settings [46]. The grid-based inputs F, M_g, M_h, V_g and V_h are concatenated and processed via a series of K Swin Transformer blocks $\{\mathcal{SW}_i\}_{i=1}^K$ [30]. After each block, we introduce a FiLM layer [39], which modulates the Swin block's output conditioned on z_C:

$$z_{i+1} \ = \ \mathrm{FiLM}_i\big(\mathcal{SW}_i(z_i),\ z_C\big), \quad z_i \in \mathbb{R}^{H_i \times W_i \times C_i} \tag{3}$$

Inspired by Swin Transformer U-Net architectures [4, 11], we retain all intermediate latent representations $\{z_i\}_{i=1}^K$ to keep information at various semantic levels. To enable information aggregation, we flatten the spatial dimensions H_i and W_i and use Multi-Head Attention Pooling [25, 54], where a set of trainable query vectors $\{q_i\}_{i=1}^K, q_i \in \mathbb{R}^{T \times C_i}$ is defined. T corresponds to the number of weight and bias tensors in the target PINN. The queries q_i are then used in a multi-head attention module together with z_i reshaped into $kv_i \in \mathbb{R}^{H_i \times W_i \times C_i}$:

$$p_i = \mathrm{MultiHeadAttention}_i(q_i, kv_i, kv_i), \quad p_i \in \mathbb{R}^{T \times C_i}. \tag{4}$$

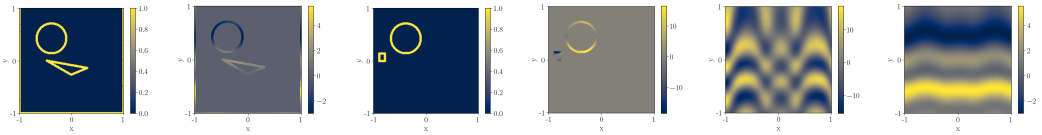
The outputs $\{p_i\}_{i=1}^K$ are concatenated along the channel dimension to produce a unified latent matrix $p = [p_1 \parallel p_2 \parallel \cdots \parallel p_K] \in \mathbb{R}^{T \times \left(\sum_{i=1}^K C_i\right)}$, containing an entry of aggregated information for each weight matrix and bias vector in the target PINN. Finally, dedicated MLPs project each entry into the required dimensionality for the corresponding weight matrix or bias vector.

We define the architecture of the target PINN as an MLP with Fourier feature mapping [44], which, when concatenated to the input (x, y), results in a dimensionality of $2N + 2$, and multiplicative skip connections [45]. Fourier encodings provide spectral expressivity for modeling high-frequency components [46], while the skip connections enhance gradient propagation and, in the context of hypernetworks, have the additional benefit of enabling dynamic depth modulation based on PDE complexity by allowing the hypernet to mask some layers.

For each PDE instance, the hypernetwork therefore generates the following parameter set $\theta^\star$:

$$\{W_0, U, V, b_0, b_u, b_v\}, \quad \{W_i, b_i\}_{i=1}^{T-2}, \quad W_{\text{out}}, b_{\text{out}}, \tag{5}$$

where d denotes the width of the latent layers. The parameter dimensions are as follows: $W_0, U, V \in \mathbb{R}^{d \times (2N+2)}$ and $b_0, b_u, b_v \in \mathbb{R}^d$; for $i = 1, \ldots, T-2$, $W_i \in \mathbb{R}^{d \times d}$ and $b_i \in \mathbb{R}^d$; and finally, $W_{\text{out}} \in \mathbb{R}^{1 \times d}$ and $b_{\text{out}} \in \mathbb{R}$. Note that we use the $\tanh$ activation function due to its bounded output space, which provides stability to the hypernet's training.



(a) $\partial\Omega_D$ (b) $g(x, y)$ (c) $\partial\Omega_N$ (d) $h(x, y)$ (e) $f(x, y)$ (f) $u(x, y)$

Figure 2: Sample generated via MMS with sampled operator $\mathcal{L}[u] = -0.31u_{xx} - 0.15u_y$ and sampled boundaries $\partial\Omega$: (a) Dirichlet boundary, (b) Dirichlet condition, (c) Neumann boundary, (d) Neumann condition, (e) source term, and (f) analytical solution.

3.3 Data Sampling

We create a synthetic dataset of PDEs by randomly drawing the differential operator $\mathcal{L}$, domain Ω, boundary data, source term f, and, when available, a reference solution u. The full dataset is a mix of two classes, supervised and unsupervised samples. For supervised samples, a manufactured analytical solution u is chosen first. We then set $f = \mathcal{L}[u]$ and derive $g(\mathbf{x}) = u(\mathbf{x})$ and / or $h(\mathbf{x}) = \frac{\partial u}{\partial n}(\mathbf{x})$ by evaluating $u(\mathbf{x})$ and its normal derivative on $\partial\Omega$. In addition to the physics-informed loss, samples of this class provide the analytical solution $u(\mathbf{x})$ as well as its derivatives that can be used for additional supervised losses during training. For unsupervised samples, the reference solution u is unknown. We sample f and boundary conditions subject to constraints designed to maximize diversity and the probability of well-posedness. Training relies solely on the physics-informed loss, as reference solutions are unavailable.

Differential operators $\mathcal{L}$ are formed by sampling $n \sim \text{Uniform}(\{1, 2, 3\})$ terms from $\mathcal{B} = \{u, u_x, u_y, u_{xx}, u_{yy}\}$ without replacement. Each selected term T_i is assigned a coefficient $c_i \sim \text{Uniform}([-2, 2])$, and the operator is defined as $\mathcal{L}[u] = \sum_{i=1}^{n} c_i T_i[u]$.

To generate supervised samples, we use MMS, an established approach for validating PDE solvers. We first construct an analytical solution $u : \Omega \to \mathbb{R}$ on a domain $\Omega \subset \mathbb{R}^2$ by applying $n \sim \text{Uniform}(\{6, \ldots, 10\})$ iterative updates starting from $u(x, y) \leftarrow 0$. Each update adds a term of the form $d \cdot \psi(ax + by + c) + e$, where $\psi \in \{x, \sin, \cos, \tanh, (1 + e^{-x})^{-1}, (1 + x^2)^{-1}\}$, and coefficients $a, b \in \{0, \text{Uniform}([-10, 10])\}$, and $c, d, e \sim \text{Uniform}([-2\pi, 2\pi])$. Terms are incorporated using one of three rules chosen uniformly at random: additive, multiplicative, or compositional.

Source term generation depends on the availability of an analytical solution. For supervised samples, we compute $f(\mathbf{x}) = \mathcal{L}[u](\mathbf{x})$ via symbolic differentiation. For unsupervised samples, where u is unknown, we set $f(\mathbf{x}) = \mathcal{N}(0, 10^2)$, i.e., a spatially constant random source drawn from a zero-mean Gaussian.

Domains $\Omega \subset [-1, 1]^2$ are generated via randomized Constructive Solid Geometry (CSG) [32]. The outer boundary $\partial\Omega_{\text{outer}}$ is defined as the unit square and may represent either purely spatial or spatio-temporal domains, with $y = -1$ marking the initial time in time-dependent PDEs. Inner boundaries $\partial\Omega_{\text{inner},i}$ are formed by subtracting randomly sampled geometric primitives (e.g., disks, polygons, rectangles) from the outer region. These boundaries enclose regions where the source term f remains active, and their role (e.g., obstacle vs. inclusion) is encoded implicitly through the boundary type.

Each inner boundary $\partial\Omega_{\text{inner},i}$ is randomly assigned Dirichlet, Neumann, or both: $u(\mathbf{x}) = g_i(\mathbf{x})$ or $\partial u/\partial n = h_i(\mathbf{x})$. For supervised samples, where $u(\mathbf{x})$ is known, we set $g = u$ and $h = \partial u/\partial n$. For unsupervised samples, boundary values are sampled to promote compatibility with the operator $\mathcal{L}[u]$. If u appears as a standalone term, we set $u = 0$ on $\partial\Omega$ to avoid trivial or inconsistent configurations (e.g., $u = f$ with nonzero f). If first-order terms (e.g., u_x, u_y) appear alone, constant Dirichlet values are used. In other cases, linear profiles are allowed, offering mild spatial variability without conflicting with the constant source term of unsupervised samples.

Despite efforts to ensure well-posedness, some unsupervised samples may still be ill-posed due to incompatible boundary and source term configurations. Nonetheless, they are essential for exposing the model to realistic complexities such as interior boundaries, inclusions, and discontinuities. These are common features in practical PDEs but are difficult to introduce through supervised data generated via MMS.

3.4 Objective Function

For each PDE instance $(\mathcal{L}, f, g, h)$ on $\Omega \subset [-1, 1]^2$ with Dirichlet ($\partial\Omega_D$) and Neumann ($\partial\Omega_N$) boundaries, HyPINO $\Phi : (\mathcal{L}, f, g, h) \mapsto \theta^\star$ produces weights $\theta^\star$ for a target PINN $u_{\theta^\star} : \Omega \to \mathbb{R}$.

$$\mathcal{J}_{\mathrm{R}} = \frac{1}{|\Omega|} \sum_{\mathbf{x}\in\Omega} \rho\left(\mathcal{L}[u_{\theta^\star}](\mathbf{x}) - f(\mathbf{x})\right) \tag{6}$$

is the residual loss and $\rho(\cdot)$ the Huber function [20]. The Dirichlet and Neumann losses are computed similarly:

$$\mathcal{J}_{\mathrm{D}} = \frac{1}{|\partial\Omega_D|} \sum_{\mathbf{x}\in\partial\Omega_D} \rho\left(u_{\theta^\star}(\mathbf{x}) - g(\mathbf{x})\right), \qquad \mathcal{J}_{\mathrm{N}} = \frac{1}{|\partial\Omega_N|} \sum_{\mathbf{x}\in\partial\Omega_N} \rho\left(\nabla u_{\theta^\star}(\mathbf{x}) \cdot \mathbf{n}(\mathbf{x}) - h(\mathbf{x})\right). \tag{7}$$

For PDEs with known analytical solutions u, we add a second-order Sobolev loss [7] that penalizes errors in function values, gradients, and second derivatives:

$$\mathcal{J}_{\mathrm{S}} = \frac{1}{|\Omega|} \sum_{\mathbf{x}\in\Omega} \sum_{k=0}^{2} \lambda_{\mathrm{S}}^{(k)} \rho\left(\nabla^k u_{\theta^\star}(\mathbf{x}) - \nabla^k u(\mathbf{x})\right). \tag{8}$$

The total loss is a weighted sum of the active terms:

$$\mathcal{J} = \lambda_{\mathrm{R}}\mathcal{J}_{\mathrm{R}} + \lambda_{\mathrm{D}}\mathcal{J}_{\mathrm{D}} + \lambda_{\mathrm{N}}\mathcal{J}_{\mathrm{N}} + \mathcal{J}_{\mathrm{S}}, \tag{9}$$

where $\mathcal{J}_{\mathrm{R}}$ is always included, $\mathcal{J}_{\mathrm{D}}$ and $\mathcal{J}_{\mathrm{N}}$ are applied when collocation points fall on $\partial\Omega_D$ or $\partial\Omega_N$, and $\mathcal{J}_{\mathrm{S}}$ is active only when the ground-truth solution u is known.

3.5 Residual–Driven Iterative Refinement

Using a hypernetwork to generate a single PINN of fixed architecture may seem restrictive, particularly in multi-physics settings where different PDEs may demand different levels of representational complexity. However, hypernetworks offer a natural mechanism for generating ensembles of PINNs at inference time, which have proven effective in reducing prediction error [1, 22, 42]. Beyond naïvely producing multiple independent samples, our framework for linear PDEs supports an ensemble construction through an iterative refinement procedure, similar in spirit to multi-stage neural networks that progressively reduce residual error [49]:

Given a PDE instance (L, f, g, h), the hypernetwork generates an initial solution $u^{(0)} := u_{\Phi(L,f,g,h)}$. We compute residuals $r_f^{(0)}$, $r_D^{(0)}$ and $r_N^{(0)}$ with respect to the PDE and boundary conditions, treat the residuals as a "delta PDE" and feed them back into the hypernetwork to obtain a corrective PINN:

$$\delta u^{(1)} := u_{\Phi(L, r_f^{(0)}, r_D^{(0)}, r_N^{(0)})}. \tag{10}$$

The updated solution is $u^{(1)} := u^{(0)} + \delta u^{(1)}$. We repeat this process for $t = 0, \ldots, T-1$:

$$u^{(t+1)} := u^{(t)} + \delta u^{(t+1)}, \quad \text{with} \quad \delta u^{(t+1)} := u_{\Phi(L, r_f^{(t)}, r_D^{(t)}, r_N^{(t)})}. \tag{11}$$

After T iterations, the final prediction is $u^{(T)} = u^{(0)} + \sum_{t=1}^{T} \delta u^{(t)}$. We refer to this model as HyPINO^i, where i defines the number of refinement rounds.

During iterative refinement, only the small PINNs are differentiated to compute residuals, whereas the hypernetwork Φ remains in inference mode. We use uniform weights for each $\delta u^{(t)}$, though adaptive weighting (e.g., scaled residual updates) remains a promising direction for future work.

4 Experiments

4.1 Training

HyPINO generates weights for a target PINN with three hidden layers and 32 hidden units per layer. The full model has 77M trainable parameters. We train the hypernetwork for 30,000 batches using the AdamW optimizer with a cosine learning rate schedule from 10^{-4} to 10^{-6} and a batch size of 128. Training was conducted on 4 NVIDIA RTX 4090 GPUs for all experiments.

Training is divided into two phases. In the first 10,000 batches, all samples are supervised with known analytical solutions, using loss weights: $\lambda_{\mathrm{R}} = 0.01$, $\lambda_{\mathrm{S}}^{(0)} = 1$, $\lambda_{\mathrm{S}}^{(1)} = 0.1$, $\lambda_{\mathrm{S}}^{(2)} = 0.01$, $\lambda_{\mathrm{D}} = 10$, and $\lambda_{\mathrm{N}} = 1$. In the remaining 20,000 batches, each batch consists of 50% supervised and 50% unsupervised samples. Loss weights are updated to: $\lambda_{\mathrm{R}} = 0.1$, $\lambda_{\mathrm{S}}^{(0)} = 1$, $\lambda_{\mathrm{S}}^{(1)} = 1$, $\lambda_{\mathrm{S}}^{(2)} = 0.1$, $\lambda_{\mathrm{D}} = 10$, and $\lambda_{\mathrm{N}} = 1$.

4.2 Baseline Models

We compare HyPINO against three baselines, each trained for 30,000 batches with batch size 128 and an initial learning rate of 10^{-4}: (i) **U-Net** [41], which shares HyPINO's encoder but replaces the hypernetwork decoder with a convolutional decoder that directly outputs a 224×224 solution grid. It is trained solely on supervised data and has 62M trainable parameters. (ii) **Poseidon** [19], a pretrained neural operator with 158M parameters. We use the Poseidon-B checkpoint and adapt it by changing the embedding and lead-time-conditioned layer normalization layers' dimensionality to match the size of our parameterization. Similarly to the U-Net, Poseidon is fine-tuned only on supervised data. (iii) **PINO** [29], a Fourier neural operator [28] with 33M parameters. We adapt it to accept 5-channel grid inputs and condition on the PDE operator using FiLM layers. It is trained using the same hybrid supervision and curriculum as HyPINO, including physics-informed losses.

4.3 Evaluation

We evaluate HyPINO and baseline models on seven standard PDE benchmarks from the PINN literature: (i) **HT** - 1D heat equation [32], (ii) **HZ** - 2D Helmholtz equation [2], (iii) **HZ-G** - Helmholtz on an irregular geometry [16], (iv) **PS-C** - Poisson with four circular interior boundaries [16], (v) **PS-L** - Poisson on an L-shaped domain [32], (vi) **PS-G** - Poisson with a Gaussian vorticity field [19], and (vii) **WV** - 1D wave equation [16]. The exact problem statements and visualizations of the corresponding parameterizations are provided in Appendix B.

Table 1: Model performance across seven PDE benchmarks. Each cell shows mean squared error (MSE) / symmetric mean absolute percentage error (SMAPE) [33]. Lower is better.

	HT	**HZ**	**HZ-G**	**PS-C**	**PS-L**	**PS-G**	**WV**
U-Net	3.5e-2 / 67	3.7e-2 / 68	6.9e-2 / 68	2.7e-2 / 33	3.9e-3 / 112	9.2e-1 / 159	3.7e-1 / 144
Poseidon	7.1e-2 / 47	3.3e-3 / 28	1.3e-1 / 65	5.3e-2 / 93	3.5e-3 / 111	7.2e-1 / 155	8.7e-1 / 138
PINO	1.4e-2 / 38	2.0e-2 / 51	6.1e-2 / 60	1.7e-1 / 65	3.3e-3 / 51	3.1e-1 / 70	3.0e-1 / 149
PINO^3	1.3e-2 / 47	7.2e-3 / 48	4.6e-2 / 64	2.8e-2 / 63	4.6e-3 / 62	2.3e-2 / 43	3.1e-1 / 127
PINO^{10}	3.9e-2 / 78	5.1e-3 / 39	1.4e-1 / 75	1.1e-2 / 48	1.0e-3 / 47	1.8e-2 / 38	8.5e-1 / 139
HyPINO	2.3e-2 / 42	5.7e-3 / 36	1.3e-1 / 64	5.6e-2 / 86	**1.7e-4** / 39	1.8e-1 / 61	2.9e-1 / 150
HyPINO^3	4.9e-4 / 11	2.7e-3 / 31	**1.6e-2 / 38**	3.4e-3 / 18	1.9e-4 / **36**	6.6e-3 / 25	2.3e-1 / 134
HyPINO^{10}	**8.0e-5 / 7**	**1.6e-3 / 22**	1.9e-2 / 40	**2.3e-3 / 15**	2.7e-4 / 40	**5.0e-3 / 24**	**1.2e-1 / 96**

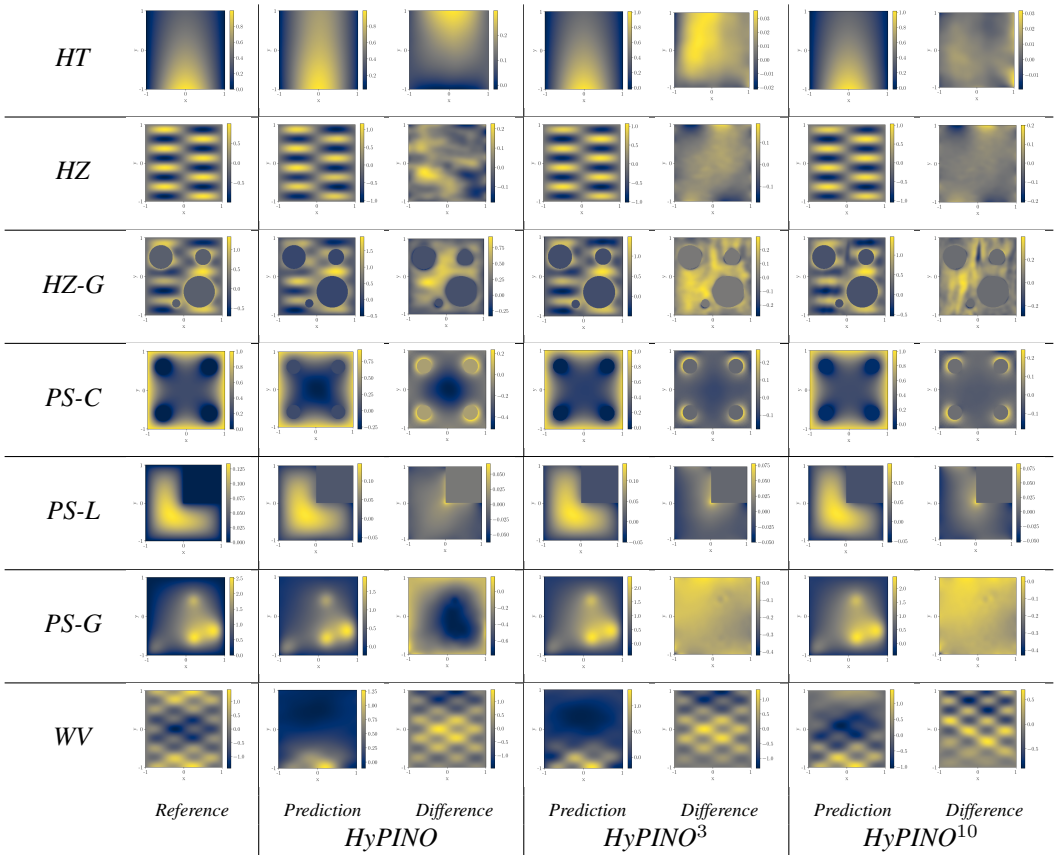
We summarize model performance across the seven PDE benchmarks in Table 1. HyPINO demonstrates consistently strong results, achieving an average rank of 2.00 across all tasks, compared to 3.00 for U-Net, 2.86 for Poseidon, and 2.14 for PINO. It is important to note that neither Poseidon nor PINO was originally designed for the PDE parameterization chosen in this study. As such, some degree of performance degradation is expected. In contrast, HyPINO contains a dedicated embedding mechanism tailored to this parameterization, but faces the challenge of operating in a significantly less structured output space compared to the grid-based outputs of the baselines. Its competitive

zero-shot performance under these conditions is therefore noteworthy. Across all benchmarks, models trained with physics-informed objectives generally outperform those relying solely on supervised data. This indicates that incorporating physics-based losses helps mitigate the generalization gap between the synthetic training data and the evaluation tasks.

Table 1 further highlights the advantages of our proposed iterative refinement approach. After three refinement iterations (HyPINO^3), we observe substantial reductions in prediction error across all but one benchmark. Notably, the MSE for PS-C and PS-G decreases by more than one order of magnitude, and an even larger improvement is observed for HT (almost two orders of magnitude). With ten refinement iterations (HyPINO^{10}), our model achieves state-of-the-art performance on all but two evaluated benchmarks, outperforming the best baseline models by factors ranging from 2.1 (on HZ against Poseidon) to 173 (HT against PINO). Table 2 shows that iterative refinement leads to a progressively more accurate prediction on the challenging WV benchmark, with the model being able to extend the undulating shape continuously further from the initial condition across the time dimension. Importantly, our results indicate that iterative refinement is not specific to HyPINO but serves as a generally effective test-time enhancement for other physics-informed neural operators, as demonstrated by the performance of PINO^3 and PINO^{10}.

We hypothesize that these improvements arise because the iterative procedure allows for correcting systematic biases introduced during training on synthetic data, which, despite its diversity and breadth, remains composed of relatively simple basis functions. As these training-induced errors tend to be consistent, artifacts produced by the initial HyPINO-generated PINNs can be systematically corrected in subsequent iterations. This residual-driven refinement yields ensembles that are significantly more effective than naive ensembles formed from independently generated target networks.

Table 2: Comparison of predictions and errors of HyPINO after zero, three, and 10 refinement rounds across all benchmark PDEs.



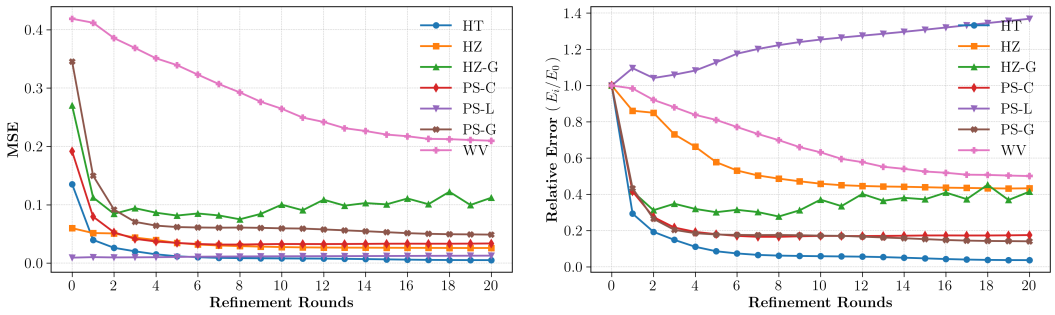


Figure 3: Effect of iterative refinement on HyPINO predictions across benchmarks. MSE (left) and relative error (right) as functions of refinement iterations. Relative error at iteration i is the ratio of MSE at iteration i to that at iteration 0.

Figure 3 illustrates these trends, showing mean squared error and relative error as functions of the number of refinement iterations. Consistent improvements are observed with additional iterations. The performance degradation on PS-L may be attributed to the already low initial error in the zeroth iteration and the small magnitudes of the solution values, resulting in correction terms that fall outside the distribution encountered during training.

4.4 Fine-tuning

The parameters $\theta^\star$ produced by HyPINO can be used to initialize PINNs for subsequent fine-tuning on specific PDE instances. We compare the convergence behavior of HyPINO-initialized PINNs with those initialized randomly and with Reptile meta-learning [37], where Reptile was trained on our synthetic dataset using 10,000 outer- and 1,000 inner-loop cycles. We also evaluate ensembles generated with HyPINO^3 and HyPINO^{10} against ensembles of equal size and architecture initialized with random weights or Reptile.

PINN fine-tuning is performed over 10,000 steps using the Adam optimizer, starting with a learning rate of 10^{-4}, decayed to 10^{-7} via a cosine schedule. Figure 4 shows convergence results on the 1D Heat Equation (HT) benchmark; results for other benchmarks and ensemble comparisons are shown in Figures 13 and 14.

HyPINO-initialized PINNs consistently start with lower loss and converge to lower final error on 4 out of 7 benchmarks. On two benchmarks, they match baseline performance, and on 1, they underperform. Quantitatively, a randomly initialized PINN requires an average of 1,068 steps to reach the initial MSE of a HyPINO-initialized model. For ensembles, matching the MSE of HyPINO^3 and HyPINO^{10} requires an average of 1,617 and 1,772 steps, respectively. Reptile-initialized PINNs converge rapidly during the first 1,000 steps, which is consistent with their meta-training configuration. However, they tend to plateau earlier and converge to higher final errors than HyPINO initializations. These findings suggest that, in addition to strong zero-shot performance, HyPINO offers a robust initialization strategy for training PINNs.

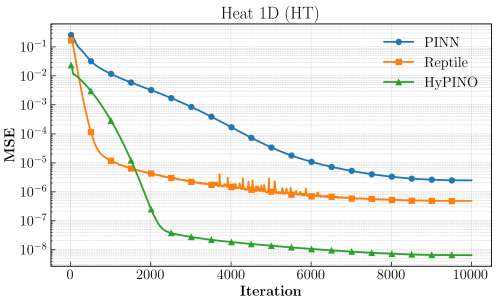


Figure 4: Convergence on the 1D Heat Equation (HT) for randomly initialized PINNs (blue), Reptile-initialized PINNs (orange), and HyPINO-initialized PINNs.

5 Conclusions & Outlook

We introduce a multi-physics neural operator based on hypernetworks (HyPINO), trained on synthetic data comprising both supervised samples, constructed using the Method of Manufactured Solutions, and purely physics-informed samples without ground-truth labels. To the best of our knowledge, our framework provides the highest degree of flexibility in the input space among existing neural operators: it accommodates variations in the differential operator, source term, domain geometry (including interior boundaries), and boundary / initial conditions. Our experiments demonstrate that training on this synthetic dataset enables strong zero-shot generalization across a diverse set of benchmark PDEs. This suggests that multi-physics neural operators can be learned with significantly reduced reliance on high-fidelity, labeled training data by leveraging synthetic datasets and self-supervised objectives. In addition, we propose a lightweight and effective iterative refinement strategy that significantly improves prediction accuracy. Notably, this refinement mechanism is generic and can be applied to other physics-informed neural operator frameworks as well. We also show that HyPINO-generated parameters provide excellent initialization for fine-tuning PINNs on specific PDE instances, yielding faster convergence and lower final errors compared to both randomly initialized and Reptile-initialized baselines.

Nonetheless, several limitations remain. Our current implementation is restricted to linear 2D PDEs with spatially uniform coefficients, which narrows the class of PDEs that HyPINO can currently address. However, the framework is inherently extensible. Future work will explore increasing the input dimensionality, incorporating spatially varying coefficients, supporting nonlinear PDEs, and modeling coupled systems. Some of these extensions may be achievable through modest modifications to the data generation process, the model's input encoding architecture, or extended training. Others may necessitate increased model capacity, either through scaling the architecture or improving the target networks' parameter generation process.

References

[1] Rafael Bischof and Michael A Kraus. Mixture-of-experts-ensemble meta-learning for physics-informed neural networks. In *Proceedings of 33rd Forum Bauinformatik*, 2022.

[2] Rafael Bischof and Michael A Kraus. Multi-objective loss balancing for physics-informed deep learning. *Computer Methods in Applied Mechanics and Engineering*, 439:117914, 2025.

[3] Lise Le Boudec, Emmanuel de Bezenac, Louis Serrano, Ramon Daniel Regueiro-Espino, Yuan Yin, and Patrick Gallinari. Learning a neural solver for parametric PDEs to enhance physics-informed methods. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=jqVj8vCQsT`.

[4] Hu Cao, Yueyue Wang, Joy Chen, Dongsheng Jiang, Xiaopeng Zhang, Qi Tian, and Manning Wang. Swin-unet: Unet-like pure transformer for medical image segmentation. In *European Conference on Computer Vision*, pages 205–218. Springer, 2022.

[5] Woojin Cho, Kookjin Lee, Donsub Rim, and Noseong Park. Hypernetwork-based meta-learning for low-rank physics-informed neural networks. *Advances in Neural Information Processing Systems*, 36:11219–11231, 2023.

[6] Woojin Cho, Minju Jo, Haksoo Lim, Kookjin Lee, Dongeun Lee, Sanghyun Hong, and Noseong Park. Extension of physics-informed neural networks to solving parameterized pdes. In *ICLR 2024 Workshop on AI4DifferentialEquations In Science*, 2024.

[7] Wojciech M Czarnecki, Simon Osindero, Max Jaderberg, Grzegorz Swirszcz, and Razvan Pascanu. Sobolev training for neural networks. *Advances in neural information processing systems*, 30, 2017.

[8] Filipe de Avila Belbute-Peres, Yi-fan Chen, and Fei Sha. Hyperpinn: Learning parameterized differential equations with physics-informed hypernetworks. *The symbiosis of deep learning and differential equations*, 690, 2021.

[9] Wenhao Ding, Qing He, Hanghang Tong, and Ping Wang. Pino-mbd: Physics-informed neural operator for solving coupled odes in multi-body dynamics. *arXiv preprint arXiv:2205.12262*, 2022.

[10] James Duvall, Karthik Duraisamy, and Shaowu Pan. Discretization-independent surrogate modeling over complex geometries using hypernetworks and implicit representations. *arXiv preprint arXiv:2109.07018*, 2021.

[11] Chi-Mao Fan, Tsung-Jung Liu, and Kuan-Hsien Liu. Sunet: Swin transformer unet for image denoising. In *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2333–2337. IEEE, 2022.

[12] Somdatta Goswami, Aniruddha Bora, Yue Yu, and George Em Karniadakis. Physics-informed deep neural operator networks. In *Machine learning in modeling and simulation: methods and applications*, pages 219–254. Springer, 2023.

[13] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

[14] Patrik Simon Hadorn. Shift-deeponet: Extending deep operator networks for discontinuous output functions. Master's thesis, ETH Zurich, Seminar for Applied Mathematics, 2022.

[15] Zhongkai Hao, Zhengyi Wang, Hang Su, Chengyang Ying, Yinpeng Dong, Songming Liu, Ze Cheng, Jian Song, and Jun Zhu. Gnot: A general neural operator transformer for operator learning. In *International Conference on Machine Learning*, pages 12556–12569. PMLR, 2023.

[16] Zhongkai Hao, Jiachen Yao, Chang Su, Hang Su, Ziao Wang, Fanzhi Lu, Zeyu Xia, Yichi Zhang, Songming Liu, Lu Lu, et al. Pinnacle: A comprehensive benchmark of physics-informed neural networks for solving pdes. *arXiv preprint arXiv:2306.08827*, 2023.

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

[18] Erisa Hasani and Rachel A Ward. Generating synthetic data for neural operators. *arXiv preprint arXiv:2401.02398*, 2024.

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

[20] Peter J Huber. Robust estimation of a location parameter. In *Breakthroughs in statistics: Methodology and distribution*, pages 492–518. Springer, 1992.

[21] Tomoharu Iwata, Yusuke Tanaka, and Naonori Ueda. Meta-learning of physics-informed neural networks for efficiently solving newly given pdes. *arXiv preprint arXiv:2310.13270*, 2023.

[22] Ameya D Jagtap and George Em Karniadakis. Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. *Communications in Computational Physics*, 28(5), 2020.

[23] Ali Kashefi and Tapan Mukerji. Physics-informed pointnet: A deep learning solver for steady-state incompressible flows and thermal fields on multiple sets of irregular geometries. *Journal of Computational Physics*, 468:111510, 2022.

[24] Jae Yong Lee, Sung Woong Cho, and Hyung Ju Hwang. Hyperdeeponet: learning operator with complex target function space using the limited resources via hypernetwork. *arXiv preprint arXiv:2312.15949*, 2023.

[25] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*, pages 3744–3753. PMLR, 2019.

[26] Shibo Li, Tao Wang, Yifei Sun, and Hewei Tang. Multi-physics simulations via coupled fourier neural operator. *arXiv preprint arXiv:2501.17296*, 2025.

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

[28] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

[29] Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. Physics-informed neural operator for learning partial differential equations. *ACM/IMS Journal of Data Science*, 1(3):1–27, 2024.

[30] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. *arXiv e-prints*, art. arXiv:2103.14030, March 2021. doi: 10.48550/arXiv.2103.14030.

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

[32] Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. Deepxde: A deep learning library for solving differential equations. *SIAM review*, 63(1):208–228, 2021.

[33] Spyros Makridakis. Accuracy measures: theoretical and practical concerns. *International journal of forecasting*, 9(4):527–529, 1993.

[34] Tanya Marwah, Ashwini Pokle, J Zico Kolter, Zachary Lipton, Jianfeng Lu, and Andrej Risteski. Deep equilibrium based neural operators for steady-state pdes. *Advances in Neural Information Processing Systems*, 36:15716–15737, 2023.

[35] Michael McCabe, Bruno Régaldo-Saint Blancard, Liam Holden Parker, Ruben Ohana, Miles Cranmer, Alberto Bietti, Michael Eickenberg, Siavash Golkar, Geraud Krawezik, Francois Lanusse, et al. Multiple physics pretraining for physical surrogate models. *arXiv preprint arXiv:2310.02994*, 2023.

[36] Rudy Morel, Jiequn Han, and Edouard Oyallon. Disco: learning to discover an evolution operator for multi-physics-agnostic prediction. *arXiv preprint arXiv:2504.19496*, 2025.

[37] Alex Nichol, Joshua Achiam, and John Schulman. On First-Order Meta-Learning Algorithms. *arXiv e-prints*, art. arXiv:1803.02999, March 2018. doi: 10.48550/arXiv.1803.02999.

[38] William Oberkampf, Frederick Blottner, and Daniel Aeschliman. Methodology for computational fluid dynamics code verification/validation. In *Fluid dynamics conference*, page 2226, 1995.

[39] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. FiLM: Visual Reasoning with a General Conditioning Layer. *arXiv e-prints*, art. arXiv:1709.07871, September 2017. doi: 10.48550/arXiv.1709.07871.

[40] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

[41] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241. Springer, 2015.

[42] Patrick Stiller, Friedrich Bethke, Maximilian Böhme, Richard Pausch, Sunna Torge, Alexander Debus, Jan Vorberger, Michael Bussmann, and Nico Hoffmann. Large-scale neural solvers for partial differential equations. In *Driving Scientific and Engineering Discoveries Through the Convergence of HPC, Big Data and AI: 17th Smoky Mountains Computational Sciences and Engineering Conference, SMC 2020, Oak Ridge, TN, USA, August 26-28, 2020, Revised Selected Papers 17*, pages 20–34. Springer, 2020.

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

[44] Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in neural information processing systems*, 33:7537–7547, 2020.

[45] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43 (5):A3055–A3081, 2021.

[46] Sifan Wang, Hanwen Wang, and Paris Perdikaris. On the eigenvector bias of fourier feature networks: From regression to solving multi-scale pdes with physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 384:113938, 2021. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2021.113938. URL https://www.sciencedirect.com/science/article/pii/S0045782521002759.

[47] Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed deeponets. *Science advances*, 7(40): eabi8605, 2021.

[48] Sifan Wang, Xinling Yu, and Paris Perdikaris. When and why pinns fail to train: A neural tangent kernel perspective. *Journal of Computational Physics*, 449:110768, 2022.

[49] Yongji Wang and Ching-Yao Lai. Multi-stage neural networks: Function approximator of machine precision. *Journal of Computational Physics*, 504:112865, 2024.

[50] Weidong Wu, Yong Zhang, Lili Hao, Yang Chen, Xiaoyan Sun, and Dunwei Gong. Physics-informed partitioned coupled neural operator for complex networks. *arXiv preprint arXiv:2410.21025*, 2024.

[51] Yibo Yang and Paris Perdikaris. Physics-informed deep generative models. *arXiv preprint arXiv:1812.03511*, 2018.

[52] Zhanhong Ye, Xiang Huang, Leheng Chen, Zining Liu, Bingyang Wu, Hongsheng Liu, Zidong Wang, and Bin Dong. Pdeformer-1: A foundation model for one-dimensional partial differential equations. *arXiv preprint arXiv:2407.06664*, 2024.

[53] Biao Yuan, He Wang, Yanjie Song, Ana Heitor, and Xiaohui Chen. High-fidelity multiphysics modelling for rapid predictions using physics-informed parallel neural operator. *arXiv preprint arXiv:2502.19543*, 2025.

[54] Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Scaling vision transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12104–12113, 2022.

[55] Li Zheng, Dennis M. Kochmann, and Siddhant Kumar. Hypercan: Hypernetwork-driven deep parameterized constitutive models for metamaterials. *Extreme Mechanics Letters*, 72: 102243, 2024. ISSN 2352-4316. doi: https://doi.org/10.1016/j.eml.2024.102243. URL https://www.sciencedirect.com/science/article/pii/S2352431624001238.

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: Yes. The abstract and introduction accurately reflect the paper's contributions. HyPINO is shown to generalize zero-shot across diverse linear PDEs using a Swin Transformer-based hypernetwork trained with mixed supervision. The iterative refinement strategy and fine-tuning results are clearly presented and experimentally validated, supporting all major claims.

   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: **Answer:** The paper discusses several limitations throughout the text, although not in a dedicated section. It acknowledges that the unsupervised data generation procedure may produce ill-posed PDE instances due to incompatible boundary and source term configurations (Section 3.3). It also notes that the method is currently limited to second-order linear PDEs in two dimensions—either one spatial and one temporal or two spatial dimensions—which restricts applicability to more complex, nonlinear, or higher-dimensional problems (Section 3). Additionally, performance degradation is observed on PS-L due to low initial error and small solution magnitudes affecting refinement efficacy (Section 4.3). Finally, the adaptation of baseline models to our PDE parameterization is mentioned as a factor influencing comparative performance (Section 4.3).

   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: [NA]

   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: In addition to the model architecture (Section 3.2) and the data sampling procedure (Section 3.3 in the main paper, the processes are discussed in even greater detail in the appendix, including training configurations and hyperparameters. Furthermore, the code for running and reproducing results is provided in the supplemental material.

   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: The training dataset is created online and the code is provided in the supplemental material. Data for the evaluation benchmarks is also included and can further be obtained from the referenced sources.

   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: Training details are provided in Section 4.1 and, in more detail, in the appendix. Furthermore, code to reproduce the results is provided in the supplemental material.

   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: Results are reported with two metrics (MSE and SMAPE) on seven benchmarks and show clear trends. Given the conceptual difference between the compared models (acknowledged in the results, Section 4), this work should not be viewed as "delta method" claiming to outperform previous work. As such, exact comparison of the reported values does not make sense.

   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: Provided in Section 4.1 and the appendix.

   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: Yes, our work complies with the NeurIPS Code of Ethics in all respects. It advances scientific understanding by introducing a generalizable neural operator for PDEs. No real-world or personal data is used; all training data is synthetic. The method poses no

foreseeable misuse risk, is reproducible using public benchmarks and standard tools, and has no societal impact on individuals or communities. Training is conducted efficiently on standard hardware.

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: The work advances scientific understanding by introducing a generalizable neural operator for PDEs commonly used in engineering. As such, no direct societal impact is expected.

    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: The work advances scientific understanding by introducing a generalizable neural operator for PDEs commonly used in engineering. Therefore, any misuse can be mostly excluded.

    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: Used libraries like DeepXDE or PINNacle are appropriately referenced in the paper and mentioned in the relevant parts of the code.

    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: The synthetic datasets can be regenerated at any time. Instructions are provided in the codebase, including how to load the trained models from the respective checkpoints. No other new assets are introduced.

    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: The paper does not involve 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: The paper does not involve crowdsourcing nor 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.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: LLM was used only for writing, editing, and formatting purposes

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

A Methodology

A.1 Neural Operator Architecture

We build on the HyperPINN framework [8] and design a neural operator based on a hypernetwork that generates the weights $\theta^\star$ of a Physics-Informed Neural Network (PINN) u_θ, conditioned on a given PDE instance. Specifically, the hypernetwork learns a mapping

$$\big(\mathbf{c},\ f,\ g,\ h\big) \longmapsto \theta^\star \quad \text{such that} \quad u_{\theta^\star} \approx u, \tag{12}$$

where $\mathbf{c}$ denotes the vector of PDE coefficients, f the interior source term, g and h the Dirichlet and Neumann boundary conditions, respectively, and u the true solution.

Grid embeddings. Each grid-valued input is first passed through a Fourier feature mapping [44], which augments the input with sinusoidal encodings using five exponentially increasing bands frequencies $= 0.1 \cdot 2^i, \quad i \in \{0, 1, 2, 3, 4\}$. This enhances the network's ability to represent high-frequency content and reduces spectral bias. The Fourier mapping layer is followed by two convolutional layers with kernel size three and strides of two. For the boundary location grids M (cf. Section 3.1), we compute four embeddings: $z_D^1,\ z_D^2,\ z_N^1,\ z_N^2$. For the boundary value grids V, we compute z_g (Dirichlet values) and z_h (Neumann values). The source term yields the embedding z_f.

We define the final spatial embedding z_G by

$$z_G = \left[z_D^1 \odot z_g + z_D^2 \parallel z_N^1 \odot z_h + z_N^2 \parallel z_f\right], \tag{13}$$

where $\odot$ denotes element-wise multiplication and $[\cdot\|\cdot\|\cdot]$ denotes concatenation along the channel dimension. This composition naturally applies spatial masking to the boundary value embeddings using the boundary location masks, ensuring that information is injected only at semantically meaningful locations.

Coefficient embedding. The vector of operator coefficients $\mathbf{c} \in \mathbb{R}^5$ is embedded into a fixed-length representation $z_C \in \mathbb{R}^{d_C}$ using a Fourier feature encoder followed by a fully connected layer.

Encoding. The grid embedding z_G is processed by a sequence of K Swin Transformer blocks $\{\mathcal{SW}_i\}_{i=1}^K$. Denoting by $z^{(i)} \in \mathbb{R}^{H_i \times W_i \times C_i}$ the output of block $\mathcal{SW}_i$ and $z^{(0)} = z_G$, we interleave each block with a FiLM modulation [39] conditioned on the coefficient embeddings z_C. Concretely, we define

$$\gamma_i(z),\ \beta_i(z)\ :\ \mathbb{R}^{d_C}\ \to\ \mathbb{R}^{C_i} \tag{14}$$

via small MLPs, and write

$$z^{(i+1)}\ =\ \gamma_i(z_C)\ \odot\ \mathcal{SW}_i\big(z_G^{(i)}\big)\ +\ \beta_i(z_C), \tag{15}$$

where "$\odot$" denotes channel-wise scaling broadcast across spatial dimensions. This design ensures that at each stage, the latent grid features are adaptively modulated by both the global operator coefficients z_C.

Inspired by Swin Transformer U-Net architectures [4, 11], we retain all intermediate latent representations from the Swin blocks $\{z^{(i)}\}_{i=1}^K$ to keep information at various semantic levels.

Pooling. To aggregate spatial information into a compact latent representation suitable for parameterizing the target PINN, we perform Multi-Head Attention Pooling [25, 54] across the flattened outputs of each Swin Transformer block. Specifically, let $z_i \in \mathbb{R}^{H_i \times W_i \times C_i}$ denote the output of the i-th FiLM-modulated Swin block. We reshape it into a sequence of tokens $kv_i \in \mathbb{R}^{H_i W_i \times C_i}$, which serve as both keys and values in the attention mechanism.

For each layer $i \in \{1, \ldots, K\}$, we define a set of T trainable query vectors $q_i \in \mathbb{R}^{T \times C_i}$, where T corresponds to the number of weight and bias tensors in the target PINN. We then compute the pooled representation via multi-head attention:

$$p_i = \mathrm{MultiHeadAttention}_i(q_i, kv_i, kv_i), \quad p_i \in \mathbb{R}^{T \times C_i}. \tag{16}$$

The pooled outputs $\{p_i\}_{i=1}^K$ are concatenated along the channel dimension to produce a unified latent matrix,

$$p = [p_1 \parallel p_2 \parallel\ \cdots\ \parallel p_K] \in \mathbb{R}^{T \times \left(\sum_{i=1}^K C_i\right)}. \tag{17}$$

This matrix p contains one latent vector per target weight or bias tensor, each embedding multi-scale information aggregated across the Swin hierarchy. To obtain the actual PINN parameters, we apply a dedicated MLP to each row of p, mapping it to the appropriate shape and dimensionality required by the corresponding weight matrix or bias vector.

Target PINN. We define the architecture of the target PINN as an MLP with Fourier feature mapping [44] and multiplicative skip connections [45]. Fourier encodings provide spectral expressivity for modeling high-frequency components [46], while the skip connections enhance gradient propagation and, in the context of hypernetworks, have the additional benefit of enabling dynamic depth modulation based on PDE complexity by masking some layers.

Given a spatial input $\mathbf{x} \in \mathbb{R}^2$, the (non-trainable) encoding is defined as:

$$\xi(\mathbf{x}) = [\sin(2\pi \mathbf{B}\mathbf{x}),\ \cos(2\pi \mathbf{B}\mathbf{x}),\ \mathbf{x}] \in \mathbb{R}^{2N+2}, \tag{18}$$

where $\mathbf{B} \in \mathbb{R}^{N\times 2}$ contains exponentially spaced frequency bands.

Following Wang et al. [45], the encoded input is projected through three parallel transformations:

$$z_0 = \tanh(W_{\text{in}}\xi + b_0), \quad z_u = \tanh(U\xi + b_u), \quad z_v = \tanh(V\xi + b_v), \tag{19}$$

where $W_{\text{in}}, U, V \in \mathbb{R}^{d\times(2N+2)}$ and $b_0, b_u, b_v \in \mathbb{R}^d$. The hidden layers of the PINN are computed via:

$$z_{i+1} = z_u \odot \tanh(W_i z_i + b_i) + z_v \odot (1 - \tanh(W_i z_i + b_i)), \quad i = 0, \ldots, T-2, \tag{20}$$

with weight matrices $W_i \in \mathbb{R}^{d\times d}$ and biases $b_i \in \mathbb{R}^d$. Note that we use the $\tanh$ activation due to its bounded output range, which prevents exploding values during the hypernetwork training.

The final prediction is obtained by a linear transformation:

$$u_\theta(\mathbf{x}) = W_{\text{out}} z_{T-1} + b_{\text{out}}, \quad W_{\text{out}} \in \mathbb{R}^{1\times d},\ b_{\text{out}} \in \mathbb{R}. \tag{21}$$

For each PDE instance, the hypernetwork therefore generates the following parameter set $\theta^\star$:

$$\{W_0, U, V, b_0, b_u, b_v\}, \quad \{W_i, b_i\}_{i=1}^{T-2}, \quad W_{\text{out}}, b_{\text{out}}, \tag{22}$$

A.2 Data Sampling

A.2.1 Classes of PDEs

We construct a synthetic dataset of PDE instances by systematically sampling the governing equations, the domain, boundary conditions, source terms, and (optionally) known solutions. Two classes of samples are considered:

Class I: Supervised PDEs We generate a set of PDEs with analytical solutions via MMS. Specifically, we sample:

1. The differential operator $\mathcal{L}$.
2. The domain Ω (along with $\partial\Omega$).
3. An analytical solution $u(\mathbf{x})$.

From the chosen solution $u(\mathbf{x})$, we then compute:

- The source term $f(\mathbf{x})$ by applying $\mathcal{L}$ to u.
- The boundary conditions $g(\mathbf{x}) = u(\mathbf{x})$ and / or $h(\mathbf{x}) = \frac{\partial u}{\partial n}(\mathbf{x})$ by evaluating $u(\mathbf{x})$ and its normal derivative on $\partial\Omega$.

In addition to the self-supervised physics-informed loss, samples of this class provide $u(\mathbf{x})$ as well as its derivatives that can be used as additional supervised losses during training.

Class II: Unsupervised PDEs In this class, the analytical solution $u(\mathbf{x})$ is not known *a priori*. We create samples by choosing:

1. The differential operator $\mathcal{L}$.
2. The domain Ω (and $\partial\Omega$).
3. The source term $f(\mathbf{x})$.
4. Boundary conditions, subject to constraints designed to maximize the probability of well-posedness.

Since the ground-truth solution is not available, samples from this class rely on the self-supervised physics-informed loss to train the model. The full dataset consists of a mix of samples from both types, with the loss containing a switch to ignore the supervised loss if no analytical solution is available.

A.2.2 Sampling Differential Operators

Considering $\mathcal{B} = \{u, u_x, u_y, u_{xx}, u_{yy}\}$ to be the set of all terms that can appear in our differential operators, we sample the number of terms $n \sim \text{Uniform}(1, 2, 3)$. We then randomly select n terms from $\mathcal{B}$ without repetition and obtain their coefficients (cf. Section 3.1) $c_i \sim \text{Uniform}([-2, 2])$. The sum of the selected terms multiplied by their respective coefficients constitutes the final differential operator.

A.2.3 Sampling or Deriving the Source Terms

The source term $f(\mathbf{x})$ is handled differently based on whether the sample has a known analytical solution. For cases with an analytical solution, $u(\mathbf{x})$ is sampled (see Section A.2.4), and the source is computed by inserting u into the differential operator. For samples without analytical solution, we set the source function to a constant $f(\mathbf{x}) = \mathcal{N}(0, 10^2)$.

A.2.4 Sampling Analytical Solutions via MMS

We generate analytical solutions $u : \Omega \to \mathbb{R}$, with $\Omega \subset \mathbb{R}^2$ and $\mathbf{x} = (x, y)$, by iteratively combining n randomly constructed terms, as detailed in Algorithm 1. The number of terms is drawn from a discrete uniform distribution, $n \sim \text{Uniform}(\{6, 7, \ldots, 10\})$. The initial solution is set to zero: $u(x, y) \leftarrow 0$.

Each term is constructed by selecting a nonlinear function ψ from a predefined library:

$$\{x, \sin, \cos, \tanh, \frac{1}{1+e^{-x}}, \frac{1}{1+x^2}\}$$

The coefficients a and b are sampled from the set $\{0, \text{Uniform}([-10, 10])\}$. The remaining coefficients c, d, e are sampled as $c, d, e \sim \text{Uniform}([-2\pi, 2\pi])$. A term is then computed as $d \cdot \psi(ax + by + c) + e$, and integrated into the current state of $u(x, y)$ using one of three randomly chosen rules:

Addition:	$u(x, y) \leftarrow u(x, y) + d \cdot \psi(ax + by + c) + e$
Multiplication:	$u(x, y) \leftarrow u(x, y) \cdot d \cdot \psi(ax + by + c) + e$
Composition:	$u(x, y) \leftarrow d \cdot \psi(a \cdot u(x, y) + c) + e$

A.2.5 Sampling Physical Domains

We employ a randomized sampling procedure based on Constructive Solid Geometry (CSG) [32] to generate complex and diverse domains.

To begin, we define the domain Ω as the bounding box $[-1, 1]^2$, representing the outer boundary $\partial\Omega_{\text{outer}}$. This initial outer region can describe a purely spatial or a spatiotemporal domain. Although we continue to use (x, y) to denote the coordinate variables, in certain PDE classes (e.g., parabolic or hyperbolic), the variable y may represent the temporal dimension, with $y = -1$ corresponding to the initial time. We then create inner boundaries $\partial\Omega_{\text{inner},i}$ $(i = 1, 2, \ldots, n)$ by randomly generating geometric shapes (e.g., triangles, polygons, disks, rectangles) and subtracting them from the outer region using CSG operations. An example of a sampled domain is shown in Figure 2.

Initialize $u(x, y) \leftarrow 0$
Sample $n \sim \text{Uniform}(\{6, 7, \ldots, 10\})$
for $i = 1$ *to* n **do**
  Sample $a \sim \{0, \text{Uniform}([-10, 10])\}$
  Sample $b \sim \{0, \text{Uniform}([-10, 10])\}$
  Sample $c, d, e \sim \text{Uniform}([-2\pi, 2\pi])$
  Randomly select $\psi(x) \in \{\sin, \cos, \tanh, \sigma, x, \phi(x)\}$
  Compute term $\leftarrow d \cdot \psi(ax + by + c) + e$
  Randomly choose combination rule:
  if *add* **then**
    $u(x, y) \leftarrow u(x, y) + \text{term}$
  else if *multiply* **then**
    $u(x, y) \leftarrow u(x, y) \cdot \text{term}$
  else if *compose* **then**
    $u(x, y) \leftarrow d \cdot \psi(a \cdot u(x, y) + c) + e$
  end
end
return $u(x, y)$

Algorithm 1: Sampling procedure for random, differentiable functions that can be used as analytical solutions with MMS.

A.2.6 Sampling Boundary Conditions

We consider two types of boundary conditions on $\partial\Omega$: Dirichlet and Neumann. Note that in our setting, the computational domain is $\Omega \subset [-1, 1]^2$.

To maximize the likelihood of obtaining well-posed PDEs, we first categorize the PDE as elliptic, parabolic, or hyperbolic. Based on this classification, the following boundary conditions are imposed on the outer boundary $\partial\Omega_{\text{outer}}$:

- **Elliptic PDEs**: Dirichlet conditions on $\partial\Omega_{\text{outer}}$:

$$u(\mathbf{x}) = g(\mathbf{x}), \quad \mathbf{x} \in \partial\Omega_{\text{outer}}. \tag{23}$$

- **Parabolic PDEs**: Interpreting y as time, the *initial condition* is enforced at $y = -1$. In addition, we impose Dirichlet conditions on the spatial boundaries:

$$u(\mathbf{x}) = g(\mathbf{x}),\ \mathbf{x} \in \partial\Omega_{\text{outer}} \setminus \{y = 1\}. \tag{24}$$

- **Hyperbolic PDEs**: Similar to the parabolic setup, we set $y = -1$ as the initial time and enforce

$$u(\mathbf{x}) = g(\mathbf{x}),\ \mathbf{x} \in \partial\Omega_{\text{outer}} \setminus \{y = 1\}, \tag{25}$$

$$\frac{\partial u}{\partial n}(\mathbf{x}) \ = \ h(\mathbf{x}),\ \mathbf{x} \in \partial\Omega_{\text{outer}} \cap \{y = -1\}. \tag{26}$$

For inner boundaries (created by subtracting geometric shapes via CSG, cf. Section A.2.5), each component $\partial\Omega_{\text{inner},i}$, with $i \in \{0, 1, \ldots, n\}$, is independently assigned either a Dirichlet or Neumann condition, or both:

$$u(\mathbf{x}) = g_i(\mathbf{x}) \quad \text{or} \quad \frac{\partial u}{\partial n}(\mathbf{x}) = h_i(\mathbf{x}), \quad \mathbf{x} \in \partial\Omega_{\text{inner},i}. \tag{27}$$

For samples from Class I (Section A.2.1), where an analytical solution $u(\mathbf{x})$ is known, boundary conditions follow directly from:

$$g(\mathbf{x}) = u(\mathbf{x}), \quad \mathbf{x} \in \partial\Omega_D \tag{28}$$

$$h(\mathbf{x}) = \frac{\partial u}{\partial n}(\mathbf{x}), \quad \mathbf{x} \in \partial\Omega_N. \tag{29}$$

Here, g and h are computed by evaluating u and its normal derivative at the relevant boundary segments (outer or inner). For samples from Class II, we set the source term $f(\mathbf{x})$ to zero on the boundary (see Section A.2.3) and sample boundary values in a manner consistent with $\mathcal{L}[u]$ to ensure that the governing equation and boundary conditions remain compatible. Specifically:

- If u appears as a standalone term in $\mathcal{L}[u]$, we set $u(\mathbf{x}) = 0$ on $\partial\Omega$.
- If first-order terms such as u_x or u_y appear stand-alone, we allow the corresponding Dirichlet boundary value $u(\mathbf{x})$ to be a random constant.
- Otherwise, we also include linear functions as possible boundary values.

Despite efforts to ensure well-posedness in the generation of synthetic, unsupervised training samples, some configurations may still result in ill-posed problems due to conflicting boundary constraints and source functions. Nevertheless, these unsupervised samples are essential for enabling the model to learn from domains with interior boundaries. While supervised samples can also include inner boundaries, they are, by construction, overconstrained: the combination of the source term and outer boundary conditions suffices to determine the analytical solution. As a result, the model primarily learns to represent and adapt to interior boundary effects through unsupervised data, where such features introduce structural variability without the aid of explicit targets. Given the importance of accurately modeling interior boundaries in practical applications, we consider this trade-off acceptable.

B Experiments

All problems are reformulated over the canonical domain $[-1,1]^2$. In particular, problems originally defined over domains $[a_x, b_x] \times [a_y, b_y]$ are mapped to $[-1,1]^2$ through affine transformations of the form: $\tilde{x} = \frac{2(x-a_x)}{b_x-a_x} - 1, \quad \tilde{y} = \frac{2(y-a_y)}{b_y-a_y} - 1$, where (x, y) are original spatial coordinates and $(\tilde{x}, \tilde{y}) \in [-1,1]^2$ are the normalized coordinates.

B.1 Heat 1D (HT)

Consider the one-dimensional heat equation:

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}, \quad x \in [0,1], \quad t \in [0,1], \tag{30}$$

where $\alpha = 0.1$ denotes the thermal diffusivity constant.

Dirichlet boundary conditions are imposed as:

$$u(0,t) = u(1,t) = 0. \tag{31}$$

The initial condition is given by a periodic (sinusoidal) function:

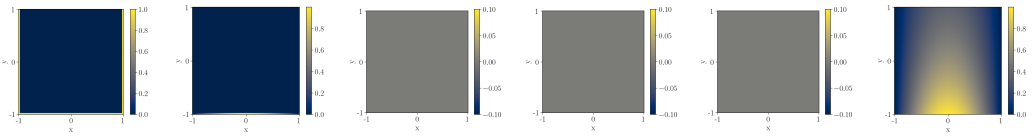
$$u(x,0) = \sin\left(\frac{n\pi x}{L}\right), \quad 0 < x < L, \quad n = 1, 2, \ldots, \tag{32}$$

where $L = 1$ is the length of the domain, and n is the frequency parameter.

The corresponding exact solution is:

$$u(x,t) = \exp\left(-\frac{n^2\pi^2\alpha t}{L^2}\right) \sin\left(\frac{n\pi x}{L}\right). \tag{33}$$

This benchmark problem is adapted from DeepXDE [32]. Figure 5 shows the parameterization of the different PDE components.



(a) $\partial\Omega_D$ (b) $g(x)$ (c) $\partial\Omega_N$ (d) $h(x)$ (e) $f(x)$ (f) $u(x)$

Figure 5: Parameterization of the 1D Heat PDE.

B.2 Helmholtz 2D (HZ)

Consider the two-dimensional Helmholtz equation:

$$\Delta u(x,y) + k^2 u(x,y) = f(x,y), \quad (x,y) \in [-1,1]^2, \tag{34}$$

where k is the wave number.

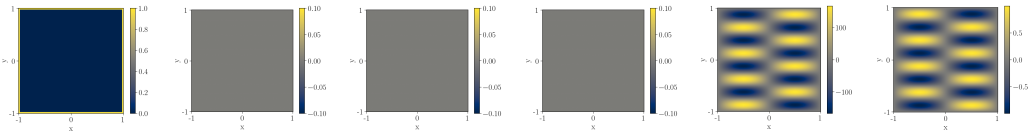
Dirichlet boundary conditions are imposed as:

$$u(-1,y) = u(1,y) = u(x,-1) = u(x,1) = 0.$$

A commonly used instance with an analytical solution is:

$$\begin{aligned} f(x,y) &= \left(-\pi^2 - (4\pi)^2 + k^2\right) \sin(\pi x) \sin(4\pi y), \\ u(x,y) &= \sin(\pi x) \sin(4\pi y). \end{aligned} \tag{35}$$

This benchmark problem is adapted from DeepXDE [32]. Figure 6 shows the parameterization of the different PDE components.



(a) $\partial\Omega_D$ (b) $g(x)$ (c) $\partial\Omega_N$ (d) $h(x)$ (e) $f(x)$ (f) $u(x)$

Figure 6: Parameterization of the 2D Helmholtz PDE.

B.3 Helmholtz 2D - Complex Geometry (HZ-G)

Consider the two-dimensional Poisson–Boltzmann (Helmholtz-type) equation:

$$-\Delta u(x,y) + k^2 u(x,y) = f(x,y), \quad (x,y) \in \Omega, \tag{36}$$

where the domain $\Omega = [-1,1]^2 \setminus \Omega_{\text{circle}}$ consists of the square $[-1,1]^2$ with four circular regions removed.

The source term is defined as:

$$f(x_1,x_2) = A \cdot \mu_2 x_2 \sin(\mu_1 \pi x_1)\sin(\mu_2 \pi x_2), \tag{37}$$

with parameters $\mu_1 = 1$, $\mu_2 = 4$, $k = 8$, and $A = 10$.

Dirichlet boundary conditions are imposed as:

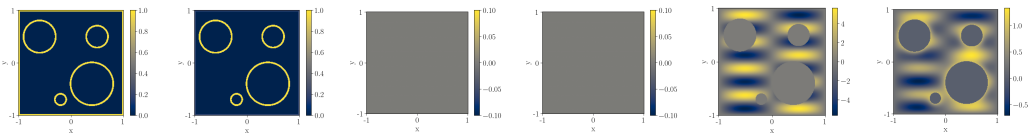
$$u(x,y) = \begin{cases} 0.2, & (x,y) \in \partial\Omega_{\text{rec}}, \\ 1.0, & (x,y) \in \partial\Omega_{\text{circle}}, \end{cases} \tag{38}$$

where $\partial\Omega_{\text{rec}}$ denotes the outer rectangular boundary, and $\partial\Omega_{\text{circle}} = \cup_{i=1}^{4} \partial R_i$ are the boundaries of the interior circles.

The circles defining the removed interior regions are given by:

$$\begin{aligned} R_1 &= \{(x,y) : (x-0.5)^2 + (y-0.5)^2 \le 0.2^2\}, \\ R_2 &= \{(x,y) : (x-0.4)^2 + (y+0.4)^2 \le 0.4^2\}, \\ R_3 &= \{(x,y) : (x+0.2)^2 + (y+0.7)^2 \le 0.1^2\}, \\ R_4 &= \{(x,y) : (x+0.6)^2 + (y-0.5)^2 \le 0.3^2\}. \end{aligned}$$

This benchmark problem is adapted from PINNacle [16]. Figure 7 shows the parameterization of the different PDE components.



(a) $\partial\Omega_D$ (b) $g(x)$ (c) $\partial\Omega_N$ (d) $h(x)$ (e) $f(x)$ (f) $u(x)$

Figure 7: Parameterization of the 2D Helmholtz-type (Poisson–Boltzmann) PDE with complex geometry.

B.4 Poisson 2D - Circles (PS-C)

Consider the two-dimensional Poisson equation:

$$-\Delta u(x,y) = 0, \quad (x,y) \in \Omega, \tag{39}$$

where the domain is defined as a rectangle with four interior circular exclusions:

$$\Omega = \Omega_{\text{rec}} \setminus \bigcup_{i=1}^{4} R_i, \quad \text{with} \quad \Omega_{\text{rec}} = [-0.5, 0.5]^2,$$

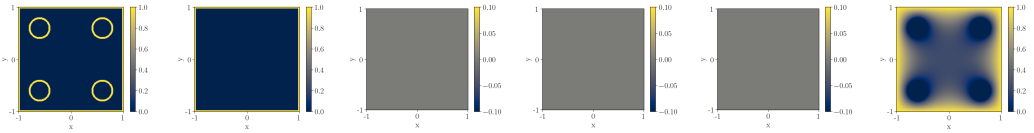
and the circular regions R_i given by:

$$\begin{aligned} R_1 &= \left\{(x,y) \;\middle|\; (x-0.3)^2 + (y-0.3)^2 \leq 0.1^2\right\}, \\ R_2 &= \left\{(x,y) \;\middle|\; (x+0.3)^2 + (y-0.3)^2 \leq 0.1^2\right\}, \\ R_3 &= \left\{(x,y) \;\middle|\; (x-0.3)^2 + (y+0.3)^2 \leq 0.1^2\right\}, \\ R_4 &= \left\{(x,y) \;\middle|\; (x+0.3)^2 + (y+0.3)^2 \leq 0.1^2\right\}. \end{aligned} \tag{40}$$

Dirichlet boundary conditions are applied as follows:

$$u(x,y) = \begin{cases} 0, & (x,y) \in \partial R_i, \\ 1, & (x,y) \in \partial \Omega_{\text{rec}}. \end{cases} \tag{41}$$

This benchmark problem is adapted from PINNacle [16]. Figure 8 shows the parameterization of the different PDE components.



(a) $\partial\Omega_D$ (b) $g(x)$ (c) $\partial\Omega_N$ (d) $h(x)$ (e) $f(x)$ (f) $u(x)$

Figure 8: Parameterization of the 2D Poisson PDE with circular inner boundaries.

B.5 Poisson 2D - L-Domain (PS-L)

Consider the two-dimensional Poisson equation:

$$-u_{xx} - u_{yy} = 1, \quad (x,y) \in \Omega, \tag{42}$$

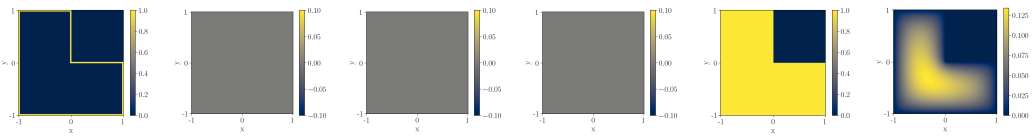
where the domain is an L-shaped region:

$$\Omega = [-1,1]^2 \setminus [0,1]^2.$$

Dirichlet boundary conditions are applied as:

$$u(x,y) = 0, \quad (x,y) \in \partial\Omega. \tag{43}$$

This benchmark problem is adapted from DeepXDE [32]. Figure 9 shows the parameterization of the different PDE components.



(a) $\partial\Omega_D$ (b) $g(x)$ (c) $\partial\Omega_N$ (d) $h(x)$ (e) $f(x)$ (f) $u(x)$

Figure 9: Parameterization of the 2D Poisson PDE on an L-shaped domain.

B.6 Poisson 2D - Gauss (PS-G)

Consider the two-dimensional Poisson equation:

$$-\Delta u(x,y) = f(x,y), \quad (x,y) \in (0,1)^2, \tag{44}$$

with homogeneous Dirichlet boundary conditions:

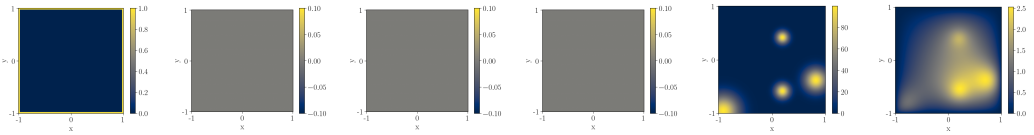
$$u(x,y) = 0, \quad (x,y) \in \partial\Omega. \tag{45}$$

The source term f is defined as a superposition of a random number N of Gaussian functions:

$$f(x,y) = \sum_{i=1}^{N} \exp\left(-\frac{(x-\mu_{x,i})^2 + (y-\mu_{y,i})^2}{2\sigma_i^2}\right), \tag{46}$$

where $N \sim \text{Geom}(0.4)$, $\mu_{x,i}, \mu_{y,i} \sim \mathcal{U}[0,1]$, and $\sigma_i \sim \mathcal{U}[0.025, 0.1]$.

We select a sample from the dataset introduced in [19]. Figure 10 shows the parameterization of the different PDE components.



(a) $\partial\Omega_D$ (b) $g(x)$ (c) $\partial\Omega_N$ (d) $h(x)$ (e) $f(x)$ (f) $u(x)$

Figure 10: Parameterization of the 2D Poisson PDE with Gaussian superposition vorticity field.

B.7 Wave 1D (WV)

Consider the one-dimensional wave equation:

$$\frac{\partial^2 u}{\partial t^2} - 4\frac{\partial^2 u}{\partial x^2} = 0, \quad (x,t) \in [0,1] \times [0,1]. \tag{47}$$

Dirichlet boundary conditions are imposed as:

$$u(0,t) = u(1,t) = 0. \tag{48}$$

The initial conditions are given by:

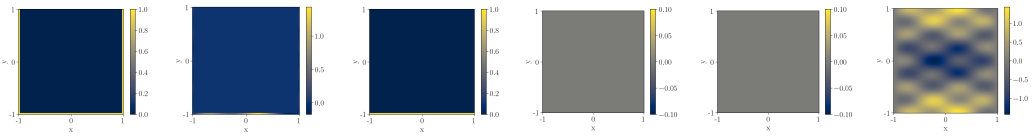
$$u(x,0) = \sin(\pi x) + \frac{1}{2}\sin(4\pi x), \tag{49}$$

$$\frac{\partial u}{\partial t}(x,0) = 0. \tag{50}$$

The corresponding exact solution is:

$$u(x,t) = \sin(\pi x)\cos(2\pi t) + \frac{1}{2}\sin(4\pi x)\cos(8\pi t). \tag{51}$$

This benchmark problem is adapted from PINNacle [16]. Figure 11 shows the parameterization of the different PDE components.



(a) $\partial\Omega_D$ (b) $g(x)$ (c) $\partial\Omega_N$ (d) $h(x)$ (e) $f(x)$ (f) $u(x)$

Figure 11: Parameterization of the 1D Wave PDE.

B.8 Baseline Models

We compare our method against three baselines. All models are trained for 30,000 batches with a batch size of 128 and an initial learning rate of 10^{-4}.

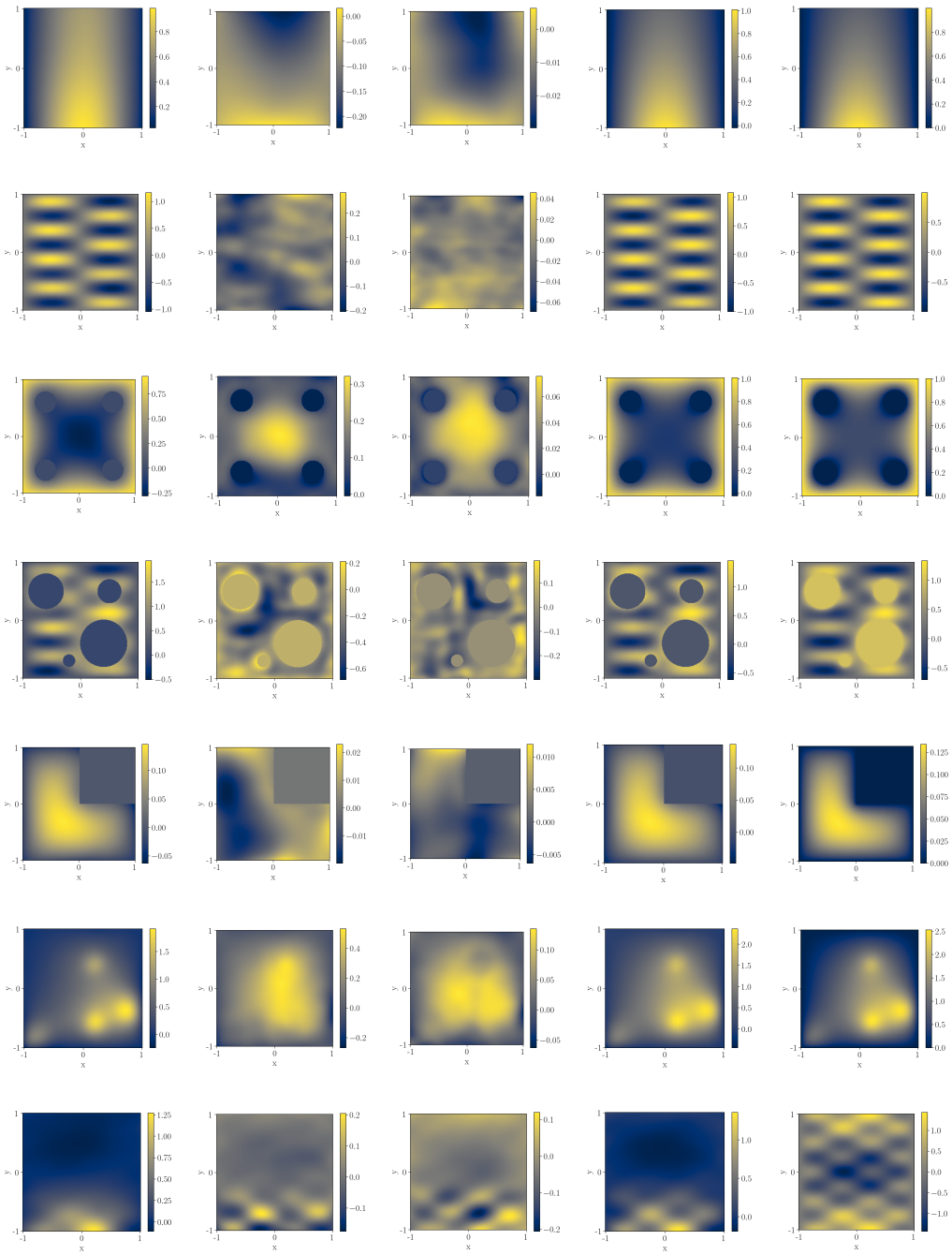
U-Net [41]. A convolutional encoder–decoder network that shares the same input encoding architecture as HyPINO, but replaces the transformer-based decoder with a purely convolutional upsampling

stack that directly outputs a solution grid of shape (224×224), matching the resolution of the input tensors. It is trained exclusively on supervised PDEs with analytical solutions, using a batch size of 128, an initial learning rate of 10^{-4}, and for 30,000 training batches. The U-Net has a total parameter count of 62M.

Poseidon [19]. A large pretrained operator network with approximately 158M parameters. We use the Poseidon-B checkpoint and adapt it by changing the input dimensionality to 5 to accept all grid-based inputs. Additionally, the lead-time-conditioned layer normalization layers—originally designed to condition on a 1D time input—are modified to condition on the 5D vector of differential operator coefficients. Poseidon is fine-tuned exclusively on supervised samples, using the same training setup as the U-Net (30,000 batches, batch size 128, initial learning rate 10^{-4}).

PINO [29]. A Fourier neural operator [28] architecture with 33M parameters, trained with joint physics-informed and supervised losses computed in Fourier space. We adapt the model to accept 5-channel grid inputs and condition on the PDE operator using FiLM layers. It follows the same hybrid supervision and training curriculum as HyPINO, including physics-informed losses, and is trained for 30,000 batches with a batch size of 128 and an initial learning rate of 10^{-4}.

B.9 Evaluation



(a) (b) (c) (d) (e)

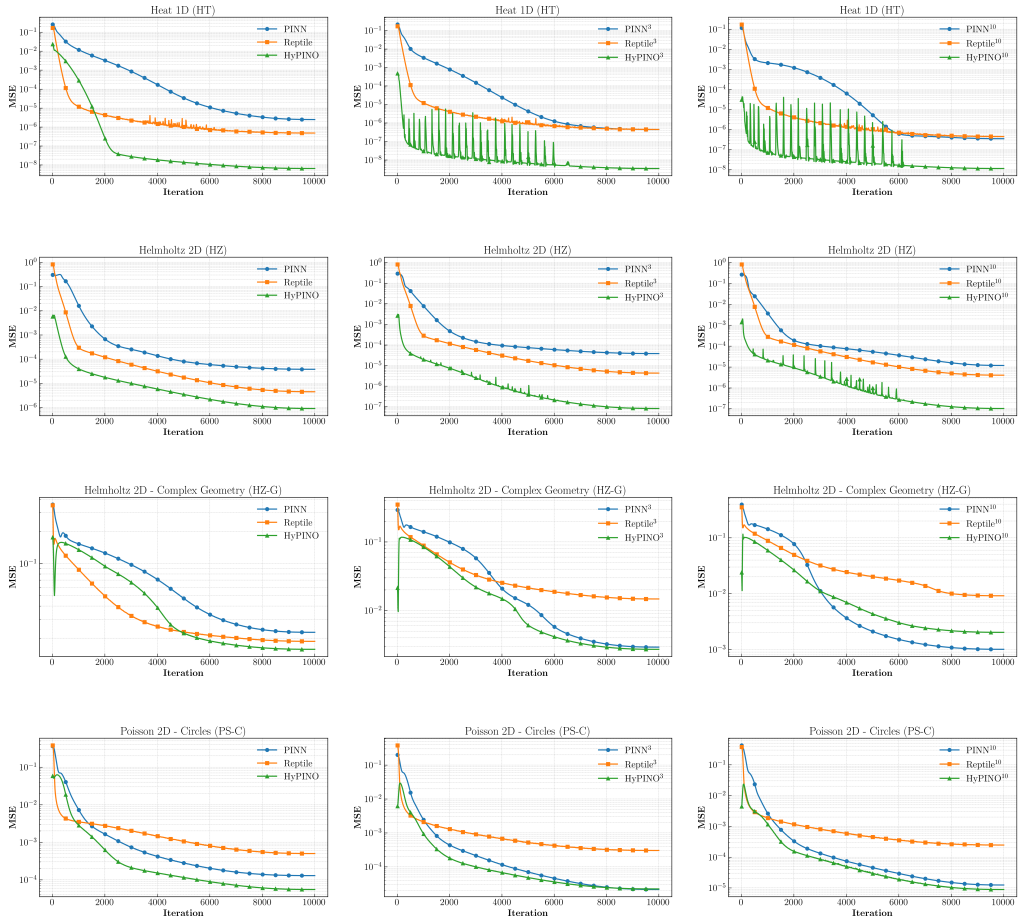
Figure 12: Visual progression of iterative refinement across different samples. Each row shows: (a) HyPINO prediction $u^{(0)}$, (b) 1st Refinement $\delta u^{(1)}$, (c) 2nd Refinement $\delta u^{(2)}$, (d) Final prediction $u^{(0)} + \delta u^{(1)} + \delta u^{(2)}$, and ground truth (e).

B.10 Fine-tuning

We investigate the utility of HyPINO-generated PINN parameters $\theta^\star$ as a prior for rapid adaptation to specific PDE instances. We compare three initialization strategies: (i) HyPINO-initialized PINNs, (ii) randomly initialized PINNs, and (iii) PINNs initialized via Reptile meta-learning [37]. For Reptile, we use 10,000 outer-loop steps on our synthetic dataset and 1,000 inner-loop updates per sample with an inner learning rate of 0.01.

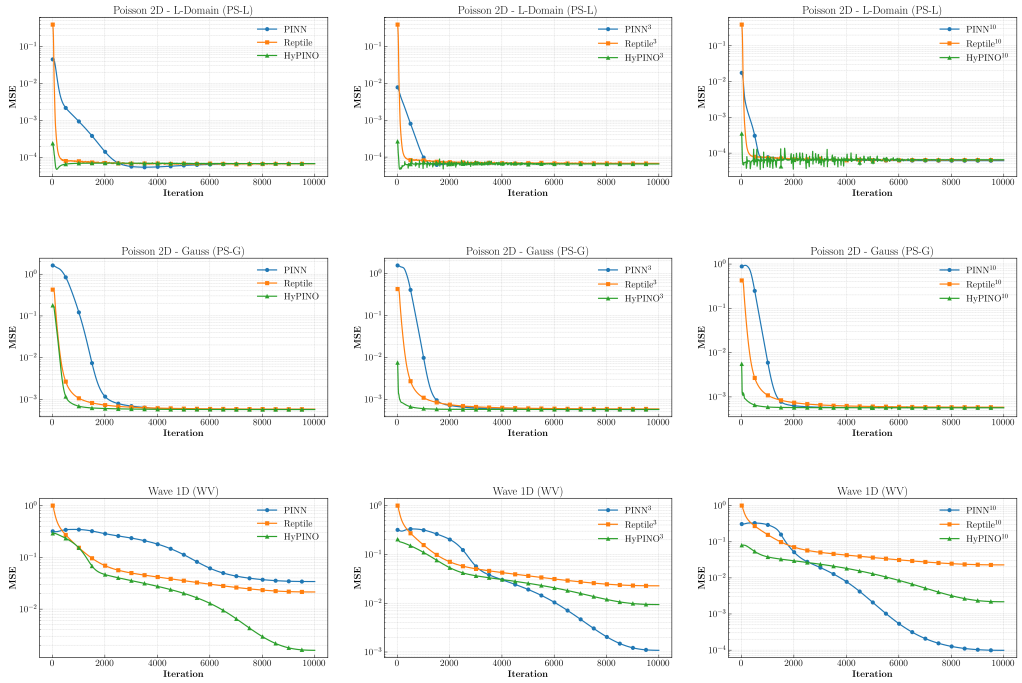
In addition to single-network performance, we evaluate the effect of initialization on ensemble-based methods. We compare ensembles of size 3 and 10 generated using HyPINO (denoted HyPINO^3 and HyPINO^{10}) against ensembles of the same size and architecture initialized either randomly or via Reptile. For the latter, we replicate the Reptile-initialized weights across all ensemble members. Note that a HyPINO^i ensemble consists of the base PINN as well as i refinement (or delta) PINNs: $u^{(0)} + \sum_{t=1}^{T} \delta u^{(t)}$, thus creating an ensemble with $i+1$ experts.

Convergence behavior across all PDE classes is reported in Figures 13 and 14.



(a) (b) (c)

Figure 13: Convergence of PINNs when fine-tuned on each of the benchmark PDE problems. We compare the convergence of different ensemble sizes: (a) single PINN, (b) ensemble of size 4 (c) ensemble of size 11, where an ensemble of size i is an ensemble of i randomly initialized PINNs (blue), i PINNs initialized via Reptile (orange), or one PINN initialized via HyPINO followed by $i-1$ refinement rounds (green).



(a) (b) (c)

Figure 14: Convergence of PINNs when fine-tuned on each of the benchmark PDE problems. We compare the convergence of different ensemble sizes: (a) single PINN, (b) ensemble of size 4 (c) ensemble of size 11, where an ensemble of size i is an ensemble of i randomly initialized PINNs (blue), i PINNs initialized via Reptile (orange), or one PINN initialized via HyPINO followed by $i - 1$ refinement rounds (green).

Resolution Invariance Ablation

Discretization-invariance is an important property for neural operators. While the output of HyPINO is a continuous PINN that can be evaluated at arbitrary spatial coordinates, the input PDE parameterization (source function and boundary masks/values) is discretized on a fixed-size grid (224×224) to match the Swin Transformer's input resolution. Following prior work [19], this limitation can be mitigated by demonstrating test-time resolution invariance when varying the input grid resolution and resizing it to (224×224).

We performed this ablation on the Helmholtz benchmark (HZ) by changing the source function resolution between 28 and 448. The results are shown in Table 3.

Table 3: Resolution invariance ablation on the Helmholtz benchmark (HZ). Each cell reports SMAPE across different input grid sizes, resized to 224×224.

	28	**56**	**96**	**112**	**140**	**168**
SMAPE	38.04	35.78	35.91	36.00	36.05	36.05
	196	**224**	**280**	**336**	**392**	**448**
SMAPE	36.05	36.04	36.05	36.03	36.04	36.04

Between resolutions of 56 and 448, SMAPE varies by less than 0.3, which indicates approximate invariance. Only at very coarse resolutions (28×28) does the performance begin to deteriorate.

L-BFGS Fine-Tuning with Different Initializations

Our initial choice to evaluate fine-tuning performance using the Adam optimizer was motivated by its wide adoption in the PINN literature. To test whether HyPINO initializations also benefit second-order optimization, we conducted additional fine-tuning experiments using L-BFGS, chosen for its broad adoption and ease of use within PyTorch. All runs used standard L-BFGS hyperparameters without tuning.

Table 4: Iterations required to match the initial MSE of a HyPINO-initialized PINN.

	HT	**HZ**	**HZ-G**	**PS-C**	**PS-L**	**PS-G**	**WV**
Random Init	4	20	N/A	36	34	11	35
Reptile Init	4	22	211	22	65	9	27

On PS-C and PS-L, Reptile requires 22 and 65 L-BFGS steps, respectively, to match HyPINO's starting error, while random initialization needs 36 and 34. On HZ-G, Reptile requires 211 steps, while random never reaches HyPINO's initial accuracy.

Table 5: Final MSE after L-BFGS fine-tuning.

	HT	**HZ**	**HZ-G**	**PS-C**	**PS-L**	**PS-G**	**WV**
Random Init	2.93e-9	**1.15e-7**	2.89e-1	3.18e-4	7.05e-5	5.69e-4	2.68e-2
Reptile Init	2.69e-9	2.18e-7	3.55e-2	9.34e-4	8.66e-5	**5.68e-4**	**3.80e-4**
HyPINO Init	**1.62e-9**	1.52e-7	**1.74e-2**	**8.19e-5**	**6.87e-5**	5.69e-4	1.94e-2

The results show that HyPINO initializations remain effective with L-BFGS. HyPINO achieves the lowest final MSE on four benchmarks (HT, PS-C, PS-L, HZ-G) and is competitive on PS-G. Only on WV does Reptile achieve the best result, while on HZ, random slightly outperforms HyPINO. These differences are especially meaningful given the high cost of L-BFGS iterations.