PICARD-INFORMED BASIS OPERATORS FOR GENERALIZABLE FEW-SHOT PDE LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural operators learn solution maps for parametric PDEs, but they need large labeled training sets and degrade under distribution shift. We introduce Picard-Informed finite Basis Operators (PIBOs), a framework that constrains operator learning with a finite basis. For linear PDEs, PIBOs trains only the responses of N fixed basis functions, then recovers any new input by exact superposition with coefficients computed in closed form rather than learned by a network. For nonlinear PDEs, it reuses those responses inside a Picard-informed loss, which removes the need for paired input–output supervision, and initializes the nonlinear operator from the pretrained linear one. Generalization error estimates bound the prediction error by the basis learning error and the truncation error. Across Poisson, reaction–diffusion, and Navier–Stokes benchmarks, PIBOs trains on 50 supervised samples where the baselines use 1,000. On the Poisson benchmark, a test-time spectral and mean shift that raises the DeepONet error to 6.51×10^{-1} leaves the PIBOs error at 6.23×10^{-3}. These gains hold within the Picard-stable regime that the analysis assumes.

1 INTRODUCTION

Partial differential equations (PDEs) model dynamic phenomena across science and engineering, from heat conduction to fluid dynamics (Anderson & Wendt, 1995; Attenborough, 2014; Landau & Lifshitz, 2013; Vallis, 2016). They range from linear forms, such as the heat (Widder, 1976) and wave equations (Langer, 1937), to nonlinear forms whose solutions may be non-unique, singular or chaotic, and are therefore harder for any solver.

Two learning paradigms have emerged as complements to classical numerical solvers. Physics-Informed Neural Networks (PINNs) (Raissi et al., 2019) embed the PDE constraint directly into the loss, which enables mesh-free learning without labeled data, but they are optimized for a single instance of the input, such as a fixed boundary or source term, and do not transfer to new inputs. Neural operators instead learn the function-to-function map $f \mapsto u$ itself. Representative frameworks include Deep Operator Networks (DeepONet) (Lu et al., 2021), Fourier Neural Operators (FNO) (Li et al., 2021), Graph Kernel Networks (GKN) (Anandkumar et al., 2020), Wavelet Neural Operators (WNO) (Tripura & Chakraborty, 2023) and Laplace Neural Operators (LNO) (Cao et al., 2024); see Liu et al. (2025) for a comprehensive review. A single training phase then serves an entire family of inputs, but the operator must be fitted on a large set of input–output pairs generated offline by a classical solver, which shifts the cost from inference to data preparation. Generalization is not assured either: the hypothesis space is effectively infinite-dimensional while the training set is finite, so sparse samples cannot regularize the learned map.

Figure 1 illustrates this failure concretely. Vanilla DeepONet requires 1,000 input functions with 100 collocation points each, that is 10^5 labeled points, for the simplest one-dimensional Poisson equation (Lu et al., 2021), and it still generalizes poorly when a test input departs from the training distribution: DeepONet predicts $v_0(x)$ accurately but misses $v_1(x)$.

We therefore ask whether the hypothesis space itself, rather than the size of the training set, should be constrained. We propose Picard-Informed finite Basis Operators (PIBOs), an architecture-agnostic framework that makes a finite basis the object the operator learns over, for both linear and nonlinear PDEs, and trains two separate operator networks, one for linear PDEs and one for nonlinear PDEs, where the latter calls the pretrained linear model through Picard iteration.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

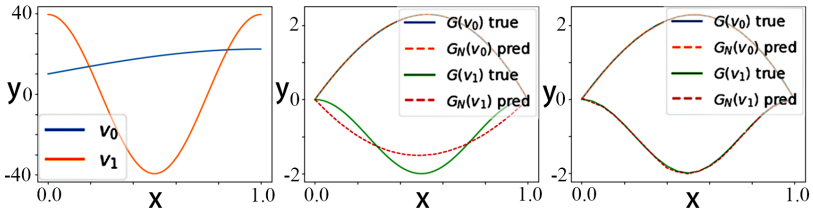

Figure 1: Illustration of Neural Operator $\mathcal{G}_N$'s generalization ability for the solution operator $\mathcal{G}$ of the 1D Poisson Equation. (**Left**) Two distinct input function $v_0(x) = \pi^2 \sin\left(\frac{\pi}{2}x\right) + 10$ and $v_1(x) = 4\pi^2 \cos(2\pi x)$. (**Middle**) DeepONet: $\mathcal{G}_N(v_0)$ is well predicted, but $\mathcal{G}_N(v_1)$ is not. (**Right**) PIBOs(ours): both $\mathcal{G}_N(v_0)$ and $\mathcal{G}_N(v_1)$ are well predicted.

Two design moves carry the framework. For linear PDEs, the operator is trained only on the responses of N fixed basis functions; because the PDE is linear, the response to any new input follows by exact superposition with coefficients computed in closed form, so no coefficient map is ever learned from data. For nonlinear PDEs, the same responses are reused inside a Picard-informed loss, which enforces the fixed-point relation $u = \mathcal{L}^{-1}(f - \mathcal{N}\mathcal{L}u)$ and removes the need for paired supervision; the nonlinear operator is initialized from the pretrained linearized operator, which supplies physics-based pseudo-labels.

The key contributions of this work are as follows:

- We introduce PIBOs, which trains on a small set of labeled basis functions once and reuses them across parametric PDEs, reducing the labeled data requirement and improving generalization.
- We propose an unsupervised Picard-informed loss and a pretrained linearized-operator initialization strategy, which turn the nonlinear learning task into an implicit Picard iteration and improve both accuracy and training efficiency.
- We provide theoretical generalization error estimates and validate PIBOs across a range of parametric linear and nonlinear PDEs, with accurate solutions and reliable generalization in the few-shot regime.

Section 3 develops the framework and states what it guarantees. Section 4 then reports the evidence in the order in which the claims are made: accuracy at matched supervision, behavior under distribution shift, comparison with the closest basis-based operator, and the ablations that isolate where the gains come from.

2 Related Work

Data-Driven Neural Operators Data-driven operator learning frameworks have proliferated in recent years (Anandkumar et al., 2020; Guibas et al., 2021; Kissas et al., 2022; Kovachki et al., 2023; Li et al., 2024; Pathak et al., 2022; Wen et al., 2023; Yue et al., 2025). Representative architectures include DeepONet (Lu et al., 2021), FNO (Li et al., 2021), Wavelet Neural Operator (Tripura & Chakraborty, 2023), and Laplace Neural Operator (Cao et al., 2024). Transformer-based solvers such as Transolver (Wu et al., 2024) use global self-attention to model PDE solution operators on general geometries. Neural Interpretable PDEs (NIPS) (Liu & Yu, 2025) combine Fourier representations with attention for scalable physics discovery, and DeepOMamba (Hu et al., 2025) applies the Mamba state-space model within the DeepONet architecture for spatio-temporal learning. Despite their strong performance, these methods require large labeled training sets and generalize poorly.

Physics-Informed Neural Operators Physics-informed neural operators reduce data requirements by embedding PDE constraints into the loss. Examples include Physics-Informed DeepONet (PI-DeepONet) (Wang et al., 2021), Physics-Informed FNO (PINO) (Li et al., 2024), and Variational Physics-informed Neural Operator (VINO) (Eshaghi et al., 2025). These methods typically rely on automatic differentiation, which can complicate training and increase cost (Wang et al.,

2022). In contrast, PIBOs adopts a Picard-informed formulation that avoids automatic differentiation for pointwise nonlinearities and requires it only for the convective term of the Navier–Stokes equation. The underlying finite basis expansion also yields an a posteriori bound on the error of the learned operator for each test input, stated in terms of the basis learning error and the truncation error.

Basis-to-Basis Learning Several neural operators incorporate basis-function expansions to reduce the hypothesis space. Finite Element Operator Network (FEONet) (Lee et al., 2025) combines deep learning with the finite element method to learn finite-element coefficients. The Spectral Neural Operator (SNO) (Fanaskov & Oseledets, 2023) introduces trigonometric and Chebyshev bases into the integral kernel. Spectral Coefficient Learning via Operator Network (SCLON) (Choi et al., 2024; Lee et al., 2026) predicts spectral expansion coefficients, and Basis-to-Basis (B2B) operator learning (Ingebrand et al., 2025) learns basis representations in both input and output spaces. These methods, however, often depend on problem-specific constructions (FEONet, SCLON), fixed spectral assumptions (SNO), or substantial labeled data (B2B), and they have been evaluated mainly in distribution. PIBOs addresses these limitations by treating finite basis functions as inductive priors in an architecture-agnostic framework. Appendix F compares PIBOs with FEONet, SNO/SCLON, and B2B conceptually, and Section 4.4 compares it with B2B empirically.

3 The PIBOs Framework

3.1 Problem Setting and Design Overview

We consider both linear and nonlinear variants of the same class of PDEs:

$$\begin{cases} \mathcal{L}u(\boldsymbol{x}) = f(\boldsymbol{x}), & \boldsymbol{x} \in \Omega, \\ \mathcal{B}u(\boldsymbol{x}) = 0, & \boldsymbol{x} \in \partial\Omega, \end{cases} \qquad \begin{cases} \mathcal{L}u(\boldsymbol{x}) + \mathcal{N}\mathcal{L}u(\boldsymbol{x}) = f(\boldsymbol{x}), & \boldsymbol{x} \in \Omega, \\ \mathcal{B}u(\boldsymbol{x}) = 0, & \boldsymbol{x} \in \partial\Omega. \end{cases} \tag{1}$$

Here $\mathcal{L}$ and $\mathcal{N}\mathcal{L}$ denote the linear and nonlinear differential operators, respectively, and $\mathcal{B}$ is the boundary or initial operator. The goal is to learn the solution operator $\mathcal{G} : f \mapsto u$ for source $f \in \mathcal{F}$ and solution $u \in \mathcal{U}$, typically Hilbert spaces defined on Ω.

Two properties of existing neural operators motivate the design (cf. Figure 1). First, the network must fit an unrestricted function-to-function map, so the labeled data requirement grows with the complexity of the input family. Second, the learned map has to extrapolate to inputs outside the training distribution. PIBOs answers both by shrinking the hypothesis space to a finite basis and by keeping the learned object as small as possible: the network learns the responses of the basis functions, while the coefficients that combine them are obtained in closed form. For nonlinear PDEs the same responses stay usable, because Picard iteration rewrites $\mathcal{L}u + \mathcal{N}\mathcal{L}u = f$ as a sequence of linear problems with a modified source term $\tilde{f}(u) = f - \mathcal{N}\mathcal{L}u$.

Figure 2 illustrates the architecture and Algorithm 1 the procedure. Section 3.2 details the linear pipeline, Section 3.3 the nonlinear one, and Section 3.4 states what the two guarantee.

Algorithm 1 Solution Procedure of PIBOs

Require: Finite basis functions $\{\hat{h}_i, \hat{u}_i\}_{i=1}^N$, source function f.
Ensure: Learned solution operator $G_\theta(f)$.
if Linear PDEs **then**
  Train $G_\theta^{(0)}$ on the basis responses with the supervised loss $\mathcal{L}^0(\theta)$ of equation 2;
  Compute the coefficients $\{\alpha_i(f)\}_{i=1}^N$ in closed form and predict via equation 3.
else
  Rewrite the PDE as $\mathcal{L}u = f - \mathcal{N}\mathcal{L}(u)$ and train $G_\theta^{(0)}$ on the corresponding basis responses;
  Initialize G_θ from the pretrained linearized operator;
  Train G_θ with the Picard-informed loss $\mathcal{L}^1(\theta)$ of equation 5 and predict by $G_\theta(f)$.
end if

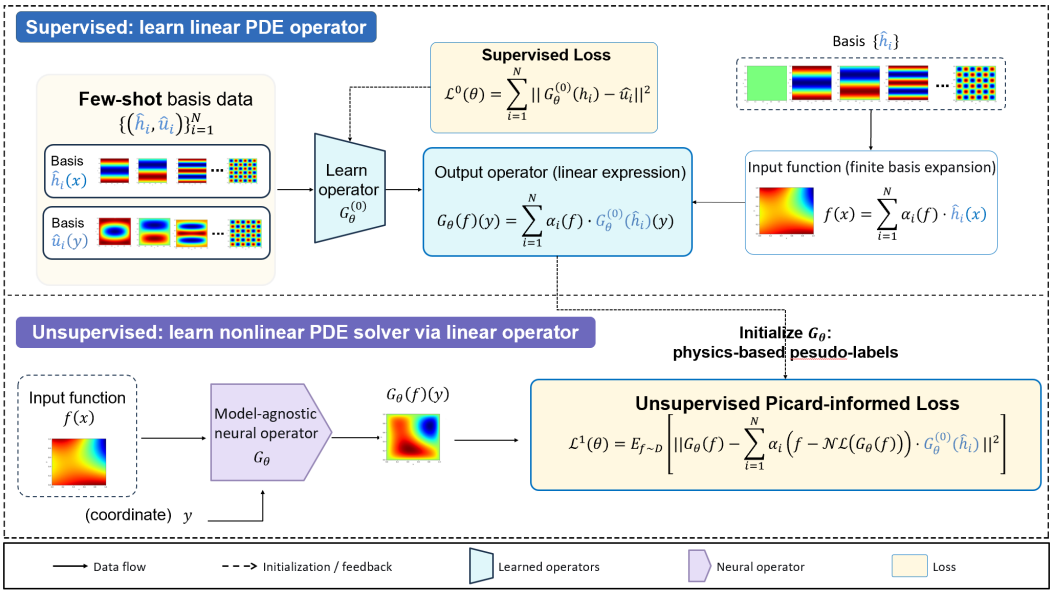

Figure 2: Schematic diagram of PIBOs. (**Top**) For linear PDEs, PIBOs learns the basis mappings and then reconstructs the solution operator by a linear combination. (**Bottom**) For nonlinear PDEs, it combines the Picard-informed loss with linearized-operator initialization for unsupervised learning.

3.2 Learning Basis Responses for Linear PDEs

Supervised stage. PIBOs trains $G_\theta^{(0)} : \hat{h}_i(x) \mapsto \hat{u}_i(y)$ for the N fixed basis functions $\{\hat{h}_i(x), \hat{u}_i(y)\}_{i=1}^N$, using the standard MSE loss

$$\mathcal{L}^0(\theta) = \frac{1}{M} \sum_{i=1}^{N} \sum_{j=1}^{M} \left| G_\theta^{(0)}(\hat{h}_i)(y_j) - \hat{u}_i(y_j) \right|^2. \tag{2}$$

The supervised set therefore has N elements however complex the input family is, which is what makes the regime few-shot.

Closed-form superposition. For an arbitrary $f(x)$, we expand it in the same N basis functions, such as the hat functions of the finite element method (Larson & Bengzon, 2013), Chebyshev, or Fourier functions: $f(x) = \sum_{i=1}^N \alpha_i \cdot \hat{h}_i(x) + \mathcal{T}_N(x)$, where $\mathcal{T}_N(x)$ is the truncation error and each α_i follows directly from the chosen basis. By linearity the solution follows as $u(y) = \sum_{i=1}^N \alpha_i \cdot \hat{u}_i(y) + \mathcal{G}(\mathcal{T}_N)(y)$, so the neural operator for linear PDEs is

$$G_\theta(f)(y) = \sum_{i=1}^{N} \alpha_i(f) \cdot G_\theta^{(0)}(\hat{h}_i)(y). \tag{3}$$

Only the N responses are learned; the recombination is exact, so no coefficient map need be extrapolated to unseen inputs.

3.3 Picard-Informed Learning for Nonlinear PDEs

For the nonlinear PDE equation 1, we write $u_\theta(y) = G_\theta(f)(y)$ for brevity. The identity $u = \mathcal{L}^{-1}(f - \mathcal{N}\mathcal{L}u)$ suggests an iterative scheme: given the current estimate $u^{(k)}$, the next iterate solves the linear problem with source $f - \mathcal{N}\mathcal{L}u^{(k)}$, reducing the nonlinear problem to a sequence of linear solves. We therefore replace f in equation 3 with $f - \mathcal{N}\mathcal{L}u_\theta$, giving

$$u_\theta(y) = \sum_{i=1}^{N} \alpha_i(f - \mathcal{N}\mathcal{L}u_\theta) \cdot G_\theta^{(0)}(\hat{h}_i)(y), \tag{4}$$

where $\alpha_i(f - \mathcal{N}\mathcal{L}u_\theta)$ is the expansion coefficient of $f - \mathcal{N}\mathcal{L}u_\theta$ under $\hat{h}_i(x)$. Because the expression is implicit, that coefficient depends on u_θ itself, which yields the unsupervised Picard-informed loss:

$$\mathcal{L}^1(f;\theta) = \frac{1}{M}\sum_{j=1}^{M}\left| u_\theta(y_j) - \sum_{i=1}^{N} \alpha_i(f - \mathcal{N}\mathcal{L}u_\theta)\cdot G_\theta^{(0)}(\hat{h}_i)(y_j)\right|^2. \tag{5}$$

The second ingredient is the initialization. Rather than starting from scratch, we initialize the nonlinear operator with the pretrained linearized operator; when $\mathcal{N}\mathcal{L} \to 0$ the explicit labels $u = \mathcal{L}^{-1}f$ are available, so this initialization supplies physics-based pseudo-labels inside the loss. The basis responses $G_\theta^{(0)}(\hat{h}_i)$ are frozen after the linear stage. The left-hand side of equation 5 is a direct forward pass on the full source f, so the loss enforces fixed-point consistency, and inference is a single forward pass with no test-time iteration; the derivatives required by $\mathcal{N}\mathcal{L}u_\theta$ for Navier–Stokes come from first-order automatic differentiation. Appendix B.3 gives the training schedule.

3.4 Why the Few-Shot Claim Holds

The two bounds below explain why the accuracy of PIBOs does not degrade with the size of the supervised set the way a generic operator does. Both are stated for the relative L^2 error at M evaluation points, formally defined in Appendix A, and proved in Appendices A.2 and A.3.

Theorem 3.1. *Assume the N basis functions form a Riesz basis in $L^2(\Omega)$, i.e., the coefficients α_i are stable and ℓ^2-bounded. Then the generalization error $\mathcal{E}^0(f;\theta)$ satisfies*

$$\mathcal{E}^0(f;\theta) \le C\left(\sqrt{\mathcal{L}^0(\theta)} + \|\mathcal{T}_N(x)\|_{L^2}\right), \tag{6}$$

where $\mathcal{L}^0$ is the training loss equation 2 for the few-shot basis functions, and the truncated error $\|\mathcal{T}_N(x)\|_{L^2} \le CN^{-s}\|f\|_{H^s}$ for a constant C independent of N and θ.

Theorem 3.2. *Under the assumptions of Theorem 3.1, and assuming the nonlinear operator $\mathcal{N}\mathcal{L}$ is Lipschitz continuous with constant $0 < L < 1$, the generalization error $\mathcal{E}^1(f;\theta)$ satisfies*

$$\mathcal{E}^1(f;\theta) \le C(1-L)^{-1}\left(\sqrt{\mathcal{L}^1(f;\theta)} + \sqrt{\mathcal{L}^0(\theta)} + \|\mathcal{T}_N(x)\|_{L^2}\right), \tag{7}$$

where $\mathcal{L}^0(\theta)$ and $\mathcal{T}_N(x)$ are as in Theorem 3.1, and $\mathcal{L}^1(f;\theta)$ is the Picard-informed loss equation 5 evaluated on the test function f.

Three features of these bounds shape the experiments that follow. First, neither bound contains a term that grows with the complexity of the input family; the error is controlled by the training loss on the N basis responses and by the truncation error N^{-s}. This is what licenses a supervised set of 50 samples: the labeled set stays at the size of the basis, and the resolution of the input family is carried by N rather than by the number of labeled pairs. Second, the truncation term decays polynomially in N while the training term decays with the fitting of the basis responses, so the two can be balanced by the choice of N — the trade-off measured in Figure 4. Third, the nonlinear bound carries the contraction constant through $(1-L)^{-1}$, so the accuracy claim holds inside the Picard-stable regime and weakens as $L \to 1$; Section 5 states where that regime ends.

4 Experiments

4.1 Setup and Evaluation Protocol

We evaluate PIBOs on three classical PDEs covering both regimes: Poisson (Poi.), reaction–diffusion (RD.) and Navier–Stokes (NS.). Input functions are sampled from periodic Gaussian random fields and the reference solutions are generated by high-fidelity numerical solvers. Training and test inputs use distinct smoothness parameters ($\alpha_{\text{train}} = 4$, $\alpha_{\text{test}} = 3.5$ unless stated otherwise), so every number below is measured under a distribution shift. PIBOs uses 50 supervised basis responses, and the comparisons are matched to that budget: supervised baselines are trained on 1,000 labeled samples and semi-supervised baselines on 50 supervised plus 1,000 unsupervised samples. The baselines are DeepONet (Lu et al., 2021), FNO (Li et al., 2021), Transolver (Wu et al., 2024), NIPS (Liu & Yu, 2025), DeepOMamba (Hu et al., 2025), PI-DeepONet (Wang et al., 2021)

Table 1: Average relative L^2 error and training time (hours) of different baselines on linear PDEs under varying training samples. The results are evaluated on 100 unseen test samples with $\alpha_{\text{test}} = 3.5$, while the supervised training data are generated with $\alpha_{\text{train}} = 4$. Bold and underlined values indicate the best and second-best results, respectively.

Eqn.	Baselines	Paradigm	20 samples		50 samples		1,000 samples	
			L^2 Err.	Time	L^2 Err.	Time	L^2 Err.	Time
Poi.	DeepONet	Sup.	9.07e-2	0.007h	6.91e-2	0.023h	1.93e-2	0.484h
	FNO	Sup.	7.04e-2	0.018h	5.88e-2	0.029h	2.30e-3	0.337h
	Transolver	Sup.	9.30e-2	0.178h	8.95e-2	0.284h	4.11e-3	3.635h
	NIPS	Sup.	<u>2.74e-2</u>	0.123h	<u>2.24e-2</u>	0.068h	5.03e-3	0.675h
	DeepOMamba	Sup.	3.48e+0	0.021h	3.12e+0	0.024h	<u>2.10e-3</u>	0.366h
	PINO	Unsup.	—	—	—	—	3.22e-2	3.011h
	PI-DeepONet	Unsup.	—	—	—	—	6.24e-2	3.855h
	PIBOs(ours)	Sup.	**1.52e-2**	0.009h	**4.43e-3**	0.022h	**5.07e-4**	0.592h
RD.	DeepONet	Sup.	4.85e-1	0.011h	1.94e-1	0.017h	2.90e-2	0.351h
	FNO	Sup.	<u>8.93e-2</u>	0.019h	<u>1.80e-2</u>	0.029h	<u>1.31e-3</u>	0.369h
	Transolver	Sup.	4.84e-1	0.199h	1.41e-1	0.326h	2.10e-3	4.165h
	NIPS	Sup.	3.15e-1	0.023h	3.06e-1	0.032h	1.01e-2	0.572h
	DeepOMamba	Sup.	2.90e+0	0.019h	3.39e+0	0.029h	6.70e-3	0.358h
	PINO	Unsup.	—	—	—	—	5.39e-2	4.933h
	PI-DeepONet	Unsup.	—	—	—	—	7.65e-2	4.261h
	PIBOs(ours)	Sup.	**3.20e-2**	0.030h	**8.40e-3**	0.075h	**1.21e-4**	0.443h
NS.	DeepONet	Sup.	6.55e-1	0.008h	4.28e-1	0.025h	4.93e-2	0.522h
	FNO	Sup.	<u>5.36e-1</u>	0.016h	<u>2.47e-1</u>	0.028h	5.40e-3	0.354h
	Transolver	Sup.	6.17e-1	0.161h	2.61e-1	0.313h	<u>4.23e-3</u>	3.828h
	NIPS	Sup.	5.52e-1	0.034h	4.52e-1	0.050h	6.75e-2	0.866h
	DeepOMamba	Sup.	3.83e+0	0.024h	1.29e+0	0.030h	2.24e-2	0.453h
	PINO	Unsup.	—	—	—	—	6.78e-2	6.356h
	PI-DeepONet	Unsup.	—	—	—	—	8.73e-2	5.891h
	PIBOs(ours)	Sup.	**1.73e-1**	0.012h	**2.64e-2**	0.060h	**2.90e-3**	1.200h

and PINO (Li et al., 2024), together with the basis-based operator B2B (Ingebrand et al., 2025). Accuracy is the relative L^2 error on 100 unseen test inputs and efficiency is training time. Fourier basis functions and a DeepONet backbone are used throughout, a choice justified in Section 4.5; the benchmark problems, input distributions, training protocol and basis constructions are given in Appendices B.1, B.2, B.3 and B.4.

4.2 Few-Shot Accuracy at Matched Supervision

Table 1 reports the linear benchmarks across training-set sizes and Table 2 the nonlinear ones at a matched budget. On the linear benchmarks PIBOs attains the lowest error in every column, with the widest margin where data are scarce: 20 supervised samples already reach 1.52×10^{-2}, below the 1.93×10^{-2} that DeepONet attains with 1,000 samples, and at 1,000 supervised samples PIBOs reaches 5.07×10^{-4}, more than $4\times$ below the best supervised baseline. On the nonlinear benchmarks PIBOs uses 50 supervised samples against the 50 plus 1,000 unsupervised of the semi-supervised baselines, and attains the lowest error of every method at that budget (Table 2), below PINO and PI-DeepONet by up to an order of magnitude. The exception is supervised Transolver, the only baseline that keeps improving as labeled pairs are added: at $20\times$ the labeled pairs and 7–$10\times$ the training time it reaches 2.11×10^{-3}, 2.71×10^{-3} and 5.10×10^{-3} on the three equations. Per labeled pair and per hour of training, PIBOs remains the more accurate method. Appendix C.1 reports the full sweep over supervision settings.

These gains are measured under the distribution shift built into the protocol, and Section 4.3 examines that shift directly.

4.3 Robustness under Distribution Shift

Training inputs use $\alpha_{\text{train}} = 4$, while test inputs use $\alpha_{\text{test}} = 2$, a substantial spectral shift. On top of that mismatch we perturb the test inputs by $f_{\text{test}} \leftarrow f_{\text{test}} + \delta$, where δ controls the magnitude

Table 2: Average relative L^2 error and training time (hours) on the nonlinear PDEs at a matched budget: the supervised baselines use 1,000 labeled samples, and the semi-supervised methods use 50 supervised plus 1,000 unsupervised samples. Each cell reports L^2 error / training time. The results are evaluated on 100 unseen test samples with $\alpha_{\text{test}} = 3.5$ and supervised training data generated with $\alpha_{\text{train}} = 4$. Bold indicates the best result within the blue-shaded semi-supervised rows, which use $20\times$ fewer labeled samples than the supervised baselines above them. The full sweep, including the 5,000-sample settings, is reported in Appendix C.1.

Method	Supervision	Poi.	RD.	NS.
DeepONet	Sup., 1,000	1.37e-2 / 0.370h	3.21e-2 / 0.385h	3.82e-2 / 0.403h
Transolver	Sup., 1,000	2.11e-3 / 3.794h	2.71e-3 / 3.648h	5.10e-3 / 3.747h
PI-DeepONet	Semi-sup., 50+1,000	5.97e-2 / 4.963h	6.34e-2 / 5.214h	9.71e-2 / 8.053h
PINO	Semi-sup., 50+1,000	4.27e-2 / 5.305h	5.88e-2 / 5.958h	8.91e-2 / 8.855h
PIBOs(ours)	Semi-sup., 50+1,000	**5.61e-3 / 0.369h**	**1.74e-2 / 0.523h**	**5.49e-2 / 0.490h**

Table 3: Mean relative L^2 error for baselines under OOD testing with mean-shift perturbations.

Method	Paradigm	#Sup.	#Unsup.	$\delta = 0.1$	$\delta = 0.2$	$\delta = 0.3$	$\delta = 0.4$
DeepONet	Sup.	1,000	0	3.55e-1	6.51e-1	8.63e-1	9.95e-1
FNO	Sup.	1,000	0	1.38e-1	2.75e-1	3.84e-1	4.53e-1
Transolver	Sup.	1,000	0	2.94e-1	5.44e-1	7.23e-1	8.32e-1
NIPS	Sup.	1,000	0	1.32e-2	2.04e-1	2.05e-1	2.06e-1
PI-DeepONet	Unsup.	0	1,000	9.45e-2	1.37e-1	1.61e-1	1.87e-1
PIBOs(ours)	Semi-sup.	50	1,000	**6.81e-3**	**6.23e-3**	**5.53e-3**	**4.74e-3**

of a global mean shift. Adding a constant changes only the lowest-order component of the input, a direction that PIBOs represents in its basis, so we treat it as a controlled diagnostic of how the DC component propagates through each method rather than as a harder distribution in itself; the spectral mismatch in the same setting supplies the genuinely out-of-distribution part.

As Figure 3 shows, PIBOs stays close to the reference solution at every perturbation level, whereas DeepONet drifts progressively. Table 3 quantifies the gap: at a mild shift ($\delta = 0.2$) the error of DeepONet grows to 6.51×10^{-1} while PIBOs remains at 6.23×10^{-3}, two orders of magnitude lower, and the ordering is unchanged across all four levels. The comparison holds the supervision budget in PIBOs's favour only in the sense that matters here — 50 supervised samples against 1,000 — and the same protocol with the spectral shift removed, together with the other shift types, is reported in Appendix D.

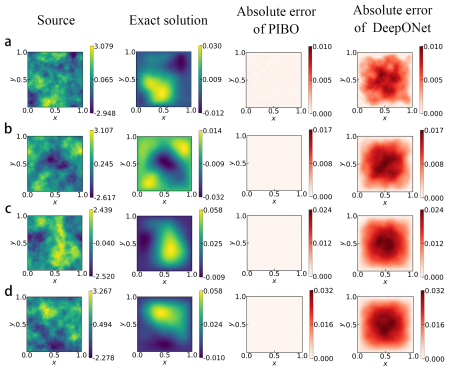

Figure 3: OOD generalization. Rows (a–d) correspond to $\delta = 0.1, 0.2, 0.3, 0.4$.

4.4 Comparison with the Closest Basis-Based Operator

Among the basis-based operator-learning methods, B2B (Ingebrand et al., 2025) is the closest to PIBOs: it likewise expands the input and output in bases, but learns the coefficient map from paired data. Table 4 compares the two under matched supervision (50 supervised samples). PIBOs reduces the relative L^2 error by a factor of 2.7–3.1 on the linear benchmarks, and the gap widens to roughly 4–6$\times$ on the nonlinear ones (Table 4). The widening is consistent with the mechanism of Section 3.3: because PIBOs computes the basis coefficients in closed form rather than learning them, the only error that must transfer to unseen inputs is the basis truncation error, whereas a learned coefficient map must additionally extrapolate from the training distribution.

Table 4: Comparison with the basis-based operator B2B (Ingebrand et al., 2025) under matched supervision. The linear rows use 50 supervised samples for both methods; the nonlinear rows use 50 supervised samples for B2B and 50 supervised plus 1,000 unsupervised samples for PIBOs.

Method	Poi.	RD.	NS.
Linear (50 supervised samples)			
B2B	1.21e-2 ± 4.5e-3	2.43e-2 ± 9.1e-3	8.16e-2 ± 2.9e-2
PIBOs(ours)	**4.43e-3 ± 2.24e-3**	**8.40e-3 ± 3.60e-3**	**2.64e-2 ± 4.70e-3**
Nonlinear (50 sup; PIBOs + 1,000 unsup)			
B2B	3.47e-2 ± 1.3e-2	7.31e-2 ± 2.7e-2	2.45e-1 ± 8.9e-2
PIBOs(ours)	**5.61e-3 ± 2.44e-3**	**1.74e-2 ± 9.81e-3**	**5.49e-2 ± 2.97e-2**

Table 5: Mean relative L^2 error for different basis functions and the usage of Fourier-layer (F).

Basis	F	Relative L^2 Error		F	Relative L^2 Error	
		Linear Poi.	Nonlinear Poi.		Linear Poi.	Nonlinear Poi.
Hat	No	8.09e-2 ± 3.72e-2	8.41e-2 ± 3.83e-2	Yes	5.14e-2 ± 2.46e-2	5.45e-2 ± 2.58e-2
Chebyshev	No	2.96e-2 ± 1.74e-2	3.36e-2 ± 1.94e-2	Yes	1.71e-2 ± 8.81e-3	2.11e-2 ± 1.04e-2
Fourier	No	1.37e-2 ± 5.82e-3	1.71e-2 ± 6.61e-3	Yes	**4.43e-3 ± 2.24e-3**	**5.61e-3 ± 2.44e-3**

4.5 What Drives the Gains

The results above establish the accuracy of PIBOs; the ablations below identify which components are responsible, by removing or replacing one at a time while holding the data and architecture fixed.

Number of basis functions. Figure 4 shows how the prediction error varies with the number of Fourier basis functions for the linear and nonlinear Poisson equations.

A richer basis improves the approximation of the operator but the gain saturates, so representation capacity has to be traded against computational cost. Accuracy is already high for a small basis, which is what makes the method efficient in the few-shot setting, and the number of basis functions does not significantly increase either the parameter count or the training time; the efficiency measurements are given in Appendix C.4.

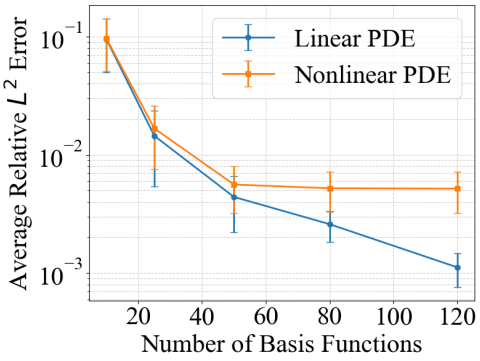

Figure 4: Relative L^2 error of PIBOs with respect to the number of Fourier basis functions on the Poisson equation.

Choice of basis and backbone. We next vary the basis function (hat, Chebyshev, Fourier) and the inclusion of a Fourier layer, which mitigates spectral bias; the constructions are specified in Appendix B.4. Table 5 shows that Fourier basis functions consistently outperform the hat and Chebyshev alternatives, particularly when paired with a Fourier layer, and this combination yields the lowest relative L^2 error in both the linear and the nonlinear setting. It is therefore adopted as the default for all experiments. The framework is agnostic to the operator it is built on: with a fixed basis, the FNO backbone reaches a comparable nonlinear Poisson error at roughly five times the training time, while the Transolver backbone is $1.5\times$ and the NIPS backbone $6\times$ less accurate; the full grid is reported in Appendix C.5. Consistent with this, equipping a network with PIBOs improves it over its fully supervised counterpart: on the linear Poisson equation the DeepONet backbone is $4\times$ more accurate with 50 supervised samples than fully supervised DeepONet is with $20\times$ that many. DeepONet offers the best accuracy–efficiency trade-off and is adopted as the backbone throughout.

Table 6: Component-wise ablation on the nonlinear benchmarks, reported as L^2 error / training time (hours). Each row removes or replaces one component of the full model; all other settings, including 50 supervised plus 1,000 unsupervised samples, are held fixed except for the pseudo-label variant, which uses no unsupervised samples. 'nan' marks a training loss that explodes.

Model or modification	Poi.	RD.	NS.
PIBOs (full, ours)	**5.61e-3 / 0.369h**	**1.74e-2 / 0.523h**	**5.49e-2 / 0.490h**
Initialization: none	6.92e-2 / 9.324h	nan	nan
Initialization: linearized PINN	5.92e-2 / 5.235h	9.93e-2 / 6.728h	4.31e-1 / 8.934h
Loss: PINN residual	5.97e-2 / 4.963h	6.34e-2 / 5.214h	9.71e-2 / 8.053h
Loss: linear pseudo-labels	1.53e-2 / 0.412h	1.12e-1 / 0.556h	5.83e-1 / 0.508h
Coefficient map: Galerkin/Picard	1.36e-2 / 0.021h	5.42e-2 / 0.054h	2.07e-1 / 0.113h

Removing each component. Table 6 removes or replaces one component of the trained model at a time. The initialization matters for both accuracy and stability: without it, training diverges on RD. and NS., and on Poi. it converges only after $25\times$ the training time, while a linearized-PINN initialization converges everywhere but stays an order of magnitude less accurate. The loss matters in the same way: replacing the Picard fixed-point consistency by a PINN residual loss, or by supervised regression to the linear pseudo-labels, degrades every equation, most sharply on the strongly nonlinear RD. and NS. cases, which indicates that the Picard iteration rather than the availability of good pseudo-labels recovers the nonlinear correction. Finally, the gains are not attributable to the finite basis alone: a classical Galerkin/Picard solver that uses the identical Fourier basis but computes the coefficients in closed form without a learned operator is uniformly less accurate, and the gap widens with nonlinearity, so the benefit comes from learning the coefficient representation rather than from the basis.

Cost of the representation. PIBOs does not buy accuracy with computation. On the nonlinear benchmarks it trains in 0.369–0.523h, against 3.648–3.794h for supervised Transolver and 4.963–8.855h for the semi-supervised baselines (Table 2). Enlarging the basis from 10 to 200 functions raises the linear-Poisson training time from 17.5s to 293.4s, while the peak GPU allocation moves by less than 15% and the parameter count stays near 122K (Appendix C.4), so the cost is paid in the input expansion rather than in the network.

5 Limitations

The nonlinear pipeline relies on a manually designed iterative scheme, and we use Picard iteration throughout, which restricts PIBOs to PDE families that admit a stable Picard reformulation. Training stability also depends on the initialization: without the linearized initialization, training diverges on two of the three nonlinear benchmarks (Table 6). The nonlinear analysis inherits the contraction condition of Theorem 3.2 and holds only in the Picard-stable regime; Appendix E.1 locates that boundary, showing that the advantage over a supervised baseline with $20\times$ more labeled data persists up to $\lambda = 5$ and reverses at $\lambda = 10$. Learning the basis functions from data would remove the manual basis choice that decides accuracy on non-periodic inputs (Appendix D.3), and Newton-type iterations may extend the method beyond the contraction regime (Appendix E.2).

6 Conclusion

We proposed Picard-Informed finite Basis Operators (PIBOs), a framework that learns the solution operator of a linear PDE from a finite set of labeled basis responses and carries over to nonlinear PDEs through a Picard-informed loss. Across the three benchmarks, PIBOs attains its reported accuracy from 50 supervised samples instead of the 1,000 used by the baselines, and retains it under spectral and mean shifts where the supervised baselines degrade by up to two orders of magnitude. Theorems 3.1 and 3.2 connect this behavior to the basis learning error, the truncation error, and the contraction property of the Picard iteration. That same contraction condition sets the scope of the claim: outside the Picard-stable regime the bound no longer holds, and automating the reformulation remains open.

AI USE STATEMENT

In this work, we used generative AI tools to assist with the writing and editing of the manuscript, namely to draft and revise portions of the paper text and to improve its readability. Following the ICLR disclosure categories, we note that these uses fall under recommended rather than required disclosure. We have not used generative AI tools for any task with required disclosure: generative AI was not used to develop the theoretical models or conceptual framework, to formulate mathematical claims, to provide ingredients for or assist in the proofs, to propose or refine hypotheses, to design the research methodology or experiments, to implement methods, to generate synthetic datasets, or to interpret results. Generative AI was likewise not used to create scientific figures or to produce software code. All AI-assisted text was reviewed and revised by the authors, and all mathematical claims, proofs, experiments, and figures were produced and verified by the authors without generative AI. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

All datasets are generated from Gaussian random fields with stated smoothness parameters; the basis constructions are detailed in Appendix B.4, the training protocol and evaluation metric in Appendix B.3, and the complete proofs in Appendices A.2 and A.3. Source code and data will be released as supplementary material to facilitate reproduction of all reported results.

REFERENCES

Anima Anandkumar, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Nikola Kovachki, Zongyi Li, Burigede Liu, and Andrew Stuart. Neural operator: Graph kernel network for partial differential equations. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020.

John David Anderson and John F. Wendt. *Computational Fluid Dynamics: The Basics with Applications*. McGraw-Hill, 1995.

Keith Attenborough. Sound propagation in the atmosphere. *Springer Handbook of Acoustics*, pp. 117–155, 2014.

Qianying Cao, Somdatta Goswami, and George Em Karniadakis. Laplace neural operator for solving differential equations. *Nature Machine Intelligence*, 6(6):631–640, 2024.

Junho Choi, Taehyun Yun, Namjung Kim, and Youngjoon Hong. Spectral operator learning for parametric pdes without data reliance. *Computer Methods in Applied Mechanics and Engineering*, 420:116678, 2024.

Mohammad Sadegh Eshaghi, Cosmin Anitescu, Manish Thombre, Yizheng Wang, Xiaoying Zhuang, and Timon Rabczuk. Variational physics-informed neural operator (VINO) for solving partial differential equations. *Computer Methods in Applied Mechanics and Engineering*, 437:117785, 2025.

Vladimir Fanaskov and Ivan Oseledets. Spectral neural operators. *Doklady Mathematics*, 108: S226–S232, 2023.

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

Zheyuan Hu, Qianying Cao, Kenji Kawaguchi, and George Em Karniadakis. DeepOMamba: State-space model for spatio-temporal PDE neural operator learning. *Journal of Computational Physics*, pp. 114272, 2025.

Tyler Ingebrand, Adam J Thorpe, Somdatta Goswami, Krishna Kumar, and Ufuk Topcu. Basis-to-basis operator learning using function encoders. *Computer Methods in Applied Mechanics and Engineering*, 435:117646, 2025.

Georgios Kissas, Jacob H Seidman, Leonardo Ferreira Guilhoto, Victor M Preciado, George J Pappas, and Paris Perdikaris. Learning operators with coupled attention. *Journal of Machine Learning Research*, 23(215):1–63, 2022.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces with applications to PDEs. *Journal of Machine Learning Research*, 24(89):1–97, 2023.

Lev Davidovich Landau and Evgenii Mikhailovich Lifshitz. *Course of theoretical physics*. Elsevier, 2013.

Rudolph E Langer. On the connection formulas and the solutions of the wave equation. *Physical Review*, 51(8):669, 1937.

Mats G Larson and Fredrik Bengzon. *The finite element method: theory, implementation, and applications*, volume 10. Springer Science & Business Media, 2013.

Jae Yong Lee, Seungchan Ko, and Youngjoon Hong. Finite element operator network for solving elliptic-type parametric PDEs. *SIAM Journal on Scientific Computing*, 47(2):C501–C528, 2025.

Myeong-Su Lee, Taehyun Yun, Youngjoon Hong, and Namjung Kim. Spectral coefficient learning via operator networks for inverse problems of parametric partial differential equations. *Engineering Applications of Artificial Intelligence*, 166:113665, 2026.

Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=c8P9NQVtmnO.

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/JMS Journal of Data Science*, 1(3):1–27, 2024.

Ning Liu and Yue Yu. Neural interpretable PDEs: Harmonizing Fourier insights with attention for scalable and interpretable physics discovery. In *International Conference on Machine Learning*, 2025.

Shengjun Liu, Yu Yu, Ting Zhang, Hanchao Liu, Xinru Liu, and Deyu Meng. Architectures, variants, and performance of neural operators: A comparative review. *Neurocomputing*, pp. 130518, 2025.

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.

Jaideep Pathak, Shashank Subramanian, Peter Harrington, Sanjeev Raja, Ashesh Chattopadhyay, Morteza Mardani, Thorsten Kurth, David Hall, Zongyi Li, Kamyar Azizzadenesheli, et al. FourCastNet: A global data-driven high-resolution weather model using adaptive Fourier neural operators. *arXiv preprint arXiv:2202.11214*, 2022.

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.

Tapas Tripura and Souvik Chakraborty. Wavelet neural operator for solving parametric partial differential equations in computational mechanics problems. *Computer Methods in Applied Mechanics and Engineering*, 404:115783, 2023.

Geoffrey K Vallis. Geophysical fluid dynamics: whence, whither and why? *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 472(2192):20160140, 2016.

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.

Sifan Wang, Hanwen Wang, and Paris Perdikaris. Improved architectures and training algorithms for deep operator networks. *Journal of Scientific Computing*, 92(2):35, 2022.

Gege Wen, Zongyi Li, Qirui Long, Kamyar Azizzadenesheli, Anima Anandkumar, and Sally M Benson. Real-time high-resolution CO_2 geological storage prediction using nested Fourier neural operators. *Energy & Environmental Science*, 16(4):1732–1741, 2023.

David Vernon Widder. *The heat equation*, volume 67. Academic Press, 1976.

Haixu Wu, Huakun Luo, Haowen Wang, Jianmin Wang, and Mingsheng Long. Transolver: A fast transformer solver for PDEs on general geometries. In *International Conference on Machine Learning*, 2024.

Xihang Yue, Yi Yang, and Linchao Zhu. DeltaPhi: Physical states residual learning for neural operators in data-limited PDE solving. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Appendix Overview

This appendix supports the main paper in six steps, ordered by the question each one answers. Appendix A states and proves the two generalization bounds used in Section 3, starting from the Picard iteration that motivates the unsupervised loss. Appendix B gives the reproducible details behind every reported number: the three benchmark PDEs and their function spaces, the input distributions, the evaluation metric, the training protocol, and the basis constructions whose truncation orders fix the size of the supervised training set. Appendix C reports the evidence that the main text only summarizes: the complete benchmark sweep, the per-sample error distributions, representative prediction fields, and the cost of the basis representation. Appendix D separates the distribution shift of the main text into its components and reports each in isolation, together with non-periodic and rougher inputs. Appendix E marks where the advantage ends—the nonlinearity strength at which the contraction condition fails and the alternatives to the Picard scheme—and extends the framework to an autoregressive task. Appendix F compares PIBOs with FEONet, SNO/SCLON, and B2B by the object each method learns from data.

Throughout, $\mathcal{G}$ denotes the exact solution operator of a linear PDE and G_θ a learned approximation. We write b_1 and b_2 for the two error sources isolated in Appendix A.2: the finite-basis approximation error and the error of the learned operator on the basis pairs. Both are reused in Appendix A.3.

A Theoretical Foundations

This section supplies the two ingredients on which the method rests: the fixed-point construction that turns a nonlinear problem into a sequence of linear ones (Appendix A.1), and the error bounds that connect the training loss to the generalization error of the learned operator (Appendix A.2 and Appendix A.3).

Error metrics. Both bounds are stated for the relative L^2 error of the learned operator at M evaluation points. For a linear PDE the error of G_θ on a test function f is

$$\mathcal{E}^0(f;\theta) = \frac{\left(\frac{1}{M}\sum_{j=1}^{M} |G_\theta(f)(y_j) - \mathcal{G}(f)(y_j)|^2\right)^{1/2}}{\left(\frac{1}{M}\sum_{j=1}^{M} |\mathcal{G}(f)(y_j)|^2\right)^{1/2}}, \tag{8}$$

and for a nonlinear PDE the error of $u_\theta = G_\theta(f)$ is

$$\mathcal{E}^1(f;\theta) = \frac{\left(\frac{1}{M}\sum_{j=1}^{M} |u_\theta(y_j) - u(y_j)|^2\right)^{1/2}}{\left(\frac{1}{M}\sum_{j=1}^{M} |u(y_j)|^2\right)^{1/2}}. \tag{9}$$

Because $\mathcal{G}$ is a bounded and boundedly invertible operator between the input and solution spaces, the two normalizing denominators are bounded above and below by constants that depend only on the PDE, so the normalization changes the bounds by a constant only. This is the metric reported in every experiment.

A.1 Picard Iteration as a Fixed-Point Construction

Picard iteration is a classical fixed-point method for nonlinear equations. Given $\mathcal{N}(u) = f$, it rewrites the problem as the fixed point $u = P(u)$ of a suitably defined operator P, and iterates $u^{(k+1)} = P(u^{(k)})$ from an initial guess $u^{(0)}$. When P is a contraction, the sequence converges to the unique fixed point u^* that solves the original problem.

Its appeal for nonlinear PDEs is that each iterate is obtained by solving a linear problem: the nonlinear terms are evaluated at the previous iterate, so the nonlinear PDE is replaced by a sequence of linear ones. In our setting this linearisation gives both the construction of the Picard-informed loss and the initialisation of the nonlinear operator from the solution operator of the linearised PDE.

For linear PDEs the solution operator is $u = \mathcal{L}^{-1}f$, where $\mathcal{L}^{-1}$ can be represented by the finite basis expansion of Section 3.2; for nonlinear PDEs we move the nonlinear part into the source term and represent the solution through the same linear operator,

$$u = \mathcal{L}^{-1}(f - \mathcal{N}\mathcal{L}u).$$

Rather than running this iteration to convergence, we use it implicitly as the training objective: the loss enforces the fixed-point relation directly, and no paired input–output supervision is required. This is the main advantage over a physics-informed residual loss, since Picard fixed-point iterations are known in numerical analysis to converge more reliably than raw residual minimisation, especially when the linear part dominates.

A.2 Proof of Theorem 3.1

Proof. Theorem 3.1 states that $\mathcal{E}^0(f;\theta) \le C(\sqrt{\mathcal{L}^0(\theta)} + \|\mathcal{T}_N\|_{L^2})$ under the Riesz-basis assumption. The proof separates the error of G_θ into the error of the learned operator on the basis pairs and the error of truncating f after N basis functions, and bounds the two terms separately.

By linearity of $\mathcal{G}$ and the basis expansion of f,

$$\mathcal{G}(f)(x_j) = \sum_{i=1}^{N} \alpha_i \cdot \mathcal{G}(\hat{h}_i)(x_j) + \mathcal{G}(\mathcal{T}_N)(x_j) = \sum_{i=1}^{N} \alpha_i \cdot \hat{u}_i(x_j) + \mathcal{G}(\mathcal{T}_N)(x_j), \tag{10}$$

whereas the learned operator is

$$G_\theta(f)(x) = \sum_{i=1}^{N} \alpha_i(f) \cdot G_\theta^{(0)}(\hat{h}_i)(x). \tag{11}$$

Subtracting equation 11 from equation 10 gives

$$G_\theta(f)(x_j) - \mathcal{G}(f)(x_j) = \underbrace{\sum_{i=1}^{N} \alpha_i \cdot \left[G_\theta^{(0)}(\hat{h}_i)(x_j) - \hat{u}_i(x_j) \right]}_{b_1} - \underbrace{\mathcal{G}(\mathcal{T}_N)(x_j)}_{b_2},$$

so that b_1 is controlled by the training loss on the basis pairs and b_2 is the contribution of the truncation error. Using $(a-b)^2 \le 2a^2 + 2b^2$ and then Cauchy–Schwarz,

$$\begin{aligned} |G_\theta(f)(x_j) - \mathcal{G}(f)(x_j)|^2 &\le 2 \left[\sum_{i=1}^{N} \alpha_i \cdot \left[G_\theta^{(0)}(\hat{h}_i)(x_j) - \hat{u}_i(x_j) \right] \right]^2 + 2\,|\mathcal{G}(\mathcal{T}_N)(x_j)|^2 \\ &\le C \left(\sum_{i=1}^{N} \alpha_i^2 \right) \left(\sum_{i=1}^{N} \left| G_\theta^{(0)}(\hat{h}_i)(x_j) - \hat{u}_i(x_j) \right|^2 \right) + 2\,|\mathcal{G}(\mathcal{T}_N)(x_j)|^2 . \end{aligned}$$

The Riesz-basis assumption makes the coefficients stable and ℓ^2-bounded,

$$\sum_{i=1}^{N} \alpha_i^2 \le C\|f\|^2,$$

and the solution operator satisfies $\|\mathcal{G}(\mathcal{T}_N)\|_{L^2} \le C\|\mathcal{T}_N\|_{L^2}$; both constants depend only on the PDE and the basis, and in particular not on N or θ. Averaging over the evaluation points and identifying the first term with the training loss $\mathcal{L}^0(\theta)$ of equation 2, we obtain

$$\mathcal{E}^0(f;\theta)^2 \le C\mathcal{L}^0(\theta) + C\|\mathcal{T}_N\|_{L^2}^2,$$

and therefore $\mathcal{E}^0(f;\theta) \le C(\sqrt{\mathcal{L}^0(\theta)} + \|\mathcal{T}_N\|_{L^2})$. □

A.3 Proof of Theorem 3.2

Proof. Theorem 3.2 extends the previous bound to the nonlinear case under the additional contraction assumption $0 < L < 1$ on the Lipschitz constant of $\mathcal{NL}$. Write $g(u) = \mathcal{NL}(u)$ for brevity. Applying equation 10 to the linearised source term $f - g(u)$ gives

$$u(x_j) = \sum_{i=1}^{N} \alpha_i(f - g(u)) \cdot \hat{u}_i(x_j) + \mathcal{G}(\mathcal{T}_N)(x_j). \tag{12}$$

Subtracting equation 12 from $u_\theta(x_j)$ and inserting the learned operator yields

$$\begin{aligned} u_\theta(x_j) - u(x_j) &= u_\theta(x_j) - \sum_{i=1}^{N} \alpha_i(f - g(u)) \cdot \hat{u}_i(x_j) - \mathcal{G}(\mathcal{T}_N)(x_j) \\ &\le \left| u_\theta(x_j) - \sum_{i=1}^{N} \alpha_i(f - g(u_\theta)) \cdot G_\theta^{(0)}(\hat{h}_i)(x_j) \right| \\ &\quad + \left| \sum_{i=1}^{N} \alpha_i(f - g(u_\theta)) \cdot \left(G_\theta^{(0)}(\hat{h}_i)(x_j) - \hat{u}_i(x_j) \right) \right| \\ &\quad + \left| \sum_{i=1}^{N} [\alpha_i(f - g(u_\theta)) - \alpha_i(f - g(u))] \cdot \hat{u}_i(x_j) \right| + \left| \mathcal{G}(\mathcal{T}_N)(x_j) \right|. \end{aligned}$$

The mean square of the first term is exactly the Picard-informed loss $\mathcal{L}^1(f;\theta)$ of equation 5, and the second is controlled by the proof of Theorem 3.1; the two are bounded by $C\sqrt{\mathcal{L}^1(f;\theta)}$ and $C\sqrt{\mathcal{L}^0(\theta)}$ respectively. The truncation term is bounded by $C\|\mathcal{T}_N\|_{L^2}$ as before. Only the third term requires the contraction assumption, and we control it by relating the coefficient differences to the difference in the source terms:

$$\begin{aligned} \mathcal{G}(f(x) - g(u))(x) &= \sum_{i=1}^{N} \alpha_i(f - g(u)) \cdot \hat{u}_i(x) + \mathcal{G}(\mathcal{T}_N)(x), \\ \mathcal{G}(f(x) - g(u_\theta))(x) &= \sum_{i=1}^{N} \alpha_i(f - g(u_\theta)) \cdot \hat{u}_i(x) + \mathcal{G}(\mathcal{T}_N^\theta)(x), \end{aligned}$$

where $\mathcal{T}_N$ and $\mathcal{T}_N^\theta$ are the truncation errors for u and u_θ respectively. Subtracting the two expansions gives

$$\sum_{i=1}^{N} [\alpha_i(f - g(u_\theta)) - \alpha_i(f - g(u))] \cdot \hat{u}_i(x_j) = \mathcal{G}(g(u) - g(u_\theta))(x_j) + \mathcal{G}(\mathcal{T}_N^\theta - \mathcal{T}_N)(x_j).$$

Using $\|\mathcal{G}(f)\|_{L^2} \le C\|f\|_{L^2}$ together with the Lipschitz bound $\|g(u) - g(v)\|_{L^2} \le L\|u - v\|_{L^2}$ with $0 < L < 1$, the third term is bounded by $L\mathcal{E}^1(f;\theta)$ up to constants. All four terms are bounded in L^2 norm, so the triangle inequality combines them into

$$(1 - L)\,\mathcal{E}^1(f;\theta) \le C\left(\sqrt{\mathcal{L}^1(f;\theta)} + \sqrt{\mathcal{L}^0(\theta)} + \|\mathcal{T}_N\|_{L^2} \right),$$

which is the stated inequality since $1 - L > 0$. □

B SETUP AND REPRODUCIBILITY

Everything needed to reproduce the reported numbers is collected here: the benchmark problems and their function spaces (Appendix B.1), the distributions from which the inputs are drawn (Appendix B.2), the training protocol, architectures and evaluation metric (Appendix B.3), and the constructions of the three bases, whose truncation orders determine the number of supervised samples (Appendix B.4).

B.1 BENCHMARK PROBLEMS

Details of the three partial differential equations used in the experiment are as follows.

2D Poisson Equation. We consider both linear and nonlinear two-dimensional Poisson equations defined on the spatial domain $\Omega = [0,1]^2$:

$$\begin{cases} -\Delta u = f, & \text{in } \Omega, \\ u = 0, & \text{on } \partial\Omega \end{cases} \qquad \begin{cases} -\Delta u + u^3 = f, & \text{in } \Omega, \\ u = 0, & \text{on } \partial\Omega \end{cases} \tag{13}$$

with homogeneous Dirichlet boundary conditions. Assuming $f \in L^2(\Omega;\mathbb{R})$, we seek $u \in H_0^1(\Omega;\mathbb{R})$. We aim to learn the solution operator $\mathcal{G} : f \mapsto u$.

1D Reaction–Diffusion Equation. We consider one-dimensional time-dependent reaction–diffusion equations, with the linear and nonlinear variants differing only in the source term:

$$\begin{cases} u_t = \nu_1 u_{xx} + f, & \text{in } \Omega_T, \\ u(0,t) = u(1,t) = 0, & t \in [0,T], \\ u(x,0) = 0, & x \in [0,1] \end{cases} \qquad \begin{cases} u_t = \nu_1 u_{xx} + \nu_2 u^2 + f, & \text{in } \Omega_T, \\ u(0,t) = u(1,t) = 0, & t \in [0,T], \\ u(x,0) = 0, & x \in [0,1] \end{cases} \tag{14}$$

where $\Omega_T = [0,1] \times [0,T]$, with diffusion coefficient $\nu_1 = 0.01$ and nonlinear coefficient $\nu_2 = 1.0$. Given $f \in L^2(\Omega_T;\mathbb{R})$, we seek $u \in C([0,T]; H_0^1((0,1);\mathbb{R}))$. The objective is to learn the solution operator $\mathcal{G} : f \mapsto u$.

2D Steady Navier–Stokes Equations. We consider steady-state incompressible 2D Navier–Stokes equations:

$$\begin{cases} -\nu\Delta \mathbf{u} + \nabla p = \mathbf{f}, & \text{in } \Omega, \\ \nabla \cdot \mathbf{u} = 0, & \text{in } \Omega \end{cases} \qquad \begin{cases} -\nu\Delta \mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p = \mathbf{f}, & \text{in } \Omega, \\ \nabla \cdot \mathbf{u} = 0, & \text{in } \Omega \end{cases} \tag{15}$$

defined on $\Omega = [0,1]^2$, with no-slip boundary conditions $\mathbf{u}|_{\partial\Omega} = 0$, where the viscosity coefficient is $\nu = 1$. The velocity field is denoted by $\mathbf{u} = (u, v)$, and the external force is $\mathbf{f} = (f_1, f_2)$. Given $\mathbf{f} \in L^2(\Omega;\mathbb{R}^2)$, we seek $(\mathbf{u}, p) \in [H_0^1(\Omega)]^2 \times L_0^2(\Omega)$. We aim to learn the solution operator $\mathcal{G} : \mathbf{f} \mapsto (\mathbf{u}, p)$.

B.2 INPUT DISTRIBUTIONS AND DATA GENERATION

To generate diverse and smooth source terms for the benchmark PDEs, we construct the input functions using periodic Gaussian random fields (GRFs) defined on a two-dimensional periodic domain. This choice enables us to systematically control the smoothness and frequency content of the inputs while ensuring compatibility with periodic boundary conditions.

Specifically, the Gaussian random field $f(x)$ is defined by its Fourier series representation

$$f(x) = \sum_{k \in \mathbb{Z}^2} \hat{f}_k \exp(ik \cdot x),$$

where the Fourier coefficients $\hat{f}_k$ are independent complex Gaussian random variables with zero mean. Their variances are prescribed according to a spectral density that decays with the wave number magnitude:

$$\mathbb{E}\left[|\hat{f}_k|^2\right] = \frac{1}{(1+|k|^2)^\alpha}.$$

The parameter $\alpha > 0$ controls the regularity of the generated fields: larger values of α correspond to smoother functions with rapidly decaying high-frequency components, while smaller values yield rougher fields with richer high-frequency content.

To ensure real-valued fields, we impose the conjugate symmetry condition $\hat{f}_{-k} = \overline{\hat{f}_k}$. The sampled Fourier coefficients are transformed to physical space using an inverse fast Fourier transform (IFFT), resulting in periodic realizations of the source term $f(x)$.

For all experiments, the training and testing datasets are generated using the same procedure but with different smoothness parameters, i.e., $\alpha_{\text{train}} \neq \alpha_{\text{test}}$, which introduces a controlled distributional shift between training and evaluation. This setup allows us to explicitly assess the generalization capability of different neural operators under changes in input regularity.

B.3 Training Protocol and Evaluation Metric

To comprehensively evaluate the performance of PIBOs, we employ the relative L^2 error as the primary evaluation metric. For a predicted solution $\hat{u}^{(i)}$ and the corresponding reference solution $u^{(i)}$ on the i-th test sample, both discretized on the same spatial grid, the relative L^2 error is defined as

$$\text{Rel-L}^2{}_i = \frac{\|\hat{u}^{(i)} - u^{(i)}\|_2}{\|u^{(i)}\|_2},$$

where $\|\cdot\|_2$ denotes the discrete L^2 norm over the spatial domain.

Given a test set consisting of N unseen samples, the mean relative L^2 error is computed as

$$\mu_{\text{Rel-L}^2} = \frac{1}{N}\sum_{i=1}^{N} \text{Rel-L}^2{}_i,$$

which reflects the overall prediction accuracy of the model.

To quantify the robustness and stability of the predictions across different input realizations, we further report the standard deviation of the relative L^2 error:

$$\sigma_{\text{Rel-L}^2} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \left(\text{Rel-L}^2{}_i - \mu_{\text{Rel-L}^2}\right)^2}.$$

Reporting both the mean and standard deviation provides a comprehensive assessment of model performance, capturing not only average accuracy but also the variability of predictions, which is particularly important under distributional shifts between training and testing data.

For PDEs with multiple solution components (e.g., velocity and pressure fields in the Navier–Stokes equations), the relative L^2 error is computed for each component individually and then aggregated to obtain the final reported error.

The detailed hyperparameter configurations of all baseline methods are summarized in Table 7. Note that the baselines differ significantly in model capacity because of their distinct architectural designs; we follow the standard hyperparameter settings reported in the original papers for each baseline.

Data Scaling. The solutions of the three PDEs have relatively small magnitudes, which limits effective learning through numerical precision and vanishing gradients. We therefore scale the solution values by a constant factor $scale_c$ during training, which improves numerical stability and facilitates gradient-based optimisation; all predictions are rescaled during evaluation so that the reported errors refer to the original solution scale. Alternative normalisations such as standard-score or min-max scaling led to no measurable improvement in our setting, and constant scaling preserves the relative structure of the solution fields.

Optimisation and Training Schedule. All models are trained with the Adam optimizer at an initial learning rate of 0.001. For the linear PDEs, the operator is trained for $2{,}500$ epochs on the supervised basis pairs. For the nonlinear PDEs, we then apply a further $2{,}500$ epochs with the Picard-informed loss described in Section 3.3.

Table 7: Model hyperparameters of PIBOs and all baselines on the 2D Poisson equation benchmark.

DeepONet		
Hyperparameter	**Value**	**Description**
Trainable parameters	114.63K	Total number of trainable parameters
Branch layers	$[128, 128, 128]$	Hidden layer widths of branch MLP
Trunk layers	$[64, 128, 128]$	Hidden layer widths of trunk MLP
Activation function	ReLU	Nonlinear activation function
Trunk input dim	2	Spatial coordinates (x, y)
PI-DeepONet		
Trainable parameters	25.60K	Total number of trainable parameters
Branch hidden layers	5	Number of hidden layers in the branch network
Trunk hidden layers	5	Number of hidden layers in the trunk network
Hidden width	50	Width of each hidden layer
Activation function	Tanh	Nonlinear activation function
FNO		
Trainable parameters	1196.80K	Total number of trainable parameters
Input channels	3	Coefficient field and spatial coordinates $(a(x, y), x, y)$
Fourier layers	4	Number of spectral convolution layers
Fourier modes	12×12	Truncation number of Fourier modes in each layer
Hidden width	32	Channel width of Fourier layers
Activation function	GeLU	Nonlinear activation function
PINO		
Trainable parameters	189.83K	Total number of trainable parameters
modes1/modes2	20/20	Truncation number of Fourier modes (x/y direction)
Hidden layers	$[64, 64, 64, 64, 64]$	Dimension of hidden layers (5 layers total)
Activation function	GeLU	Nonlinear activation function
Downsampling factor	7	Input downsampling factor
Transolver		
Trainable parameters	2826.95K	Total number of trainable parameters
Hidden dimension	128	Feature dimension of hidden representations
Number of layers	8	Number of Transformer layers
Attention heads	8	Number of self-attention heads per layer
NIPS		
Trainable parameters	64.96K	Total number of trainable parameters
Number of layers	4	Number of NIPS operator layers
Attention heads	1	Number of attention heads per layer
Key/Query dimension	40	Dimension of key and query embeddings (d_k)
Fourier modes	11	Truncation number of Fourier modes
PIBOs(ours)		
Trainable parameters	122.24K	Total number of trainable parameters
Branch hidden layers	3	Number of hidden layers in the branch network
Trunk hidden layers	3	Number of hidden layers in the trunk network
Hidden layer sizes	$[64, 128, 128]$	Width of each hidden layer
Activation function	ReLU	Nonlinear activation function

Network Architecture. Our DeepONet backbone consists of a branch network, which encodes the input function $f(x)$, and a trunk network, which processes spatial coordinates through a Fourier layer (Appendix B.3). Both are three-layer multilayer perceptrons with ReLU activations and layer widths $[64, 128, 128]$.

Computational Environment. All experiments were run in PyTorch with Python 3.8 and CUDA 11.4 on a single NVIDIA RTX 3080 Ti GPU with 12 GB of memory. Every reported result is the average over five random seeds. The solutions and the source terms are discretised on a uniform 101×101 spatial grid, and all models are trained and evaluated on this grid.

Fourier Layer. We adopt a Fourier layer module to encode spatial coordinates in two-dimensional domains, which improves the model's ability to capture periodic and oscillatory features in the input data. Specifically, for a 2D spatial point (x, y), the Fourier layer encodes the coordinates using a set of sinusoidal basis functions as follows:

$$\gamma(x,y) = \begin{bmatrix} 1 \\ \cos(k_x\omega_x x), \sin(k_x\omega_x x) \\ \cos(k_y\omega_y y), \sin(k_y\omega_y y) \\ \cos(k_x\omega_x x)\cos(k_y\omega_y y), \cos(k_x\omega_x x)\sin(k_y\omega_y y) \\ \sin(k_x\omega_x x)\cos(k_y\omega_y y), \sin(k_x\omega_x x)\sin(k_y\omega_y y) \end{bmatrix}^T$$

Here, $\omega_x = 2\pi/L_x$ and $\omega_y = 2\pi/L_y$ are spatial frequencies along the x- and y- directions, respectively, and $k_x \in \{1, 2, \ldots, M_x\}$, $k_y \in \{1, 2, \ldots, M_y\}$ are the Fourier mode indices. This embedding results in a total of $1 + 2(M_x + M_y) + 4(M_x M_y)$ encoded features. The constant term ensures that non-oscillatory components are also captured.

In our implementation, this Fourier encoding is embedded as a standalone module and integrated into the trunk network of DeepONet. During training, the input coordinates to the trunk network are first passed through the Fourier layer before being processed by a fully connected network. This design facilitates the learning of solutions with strong spatial frequency components, particularly in PDEs with periodic structures or boundary conditions.

This approach is particularly advantageous in fluid dynamics problems, where the solution exhibits spatially oscillatory patterns. Moreover, the use of such structured embeddings helps stabilize training and improve convergence, even when the boundary conditions are not strictly periodic. This makes the Fourier layer broadly applicable to a variety of boundary value problems, including those with Dirichlet and Neumann conditions.

B.4 Basis Constructions

B.4.1 Choice of Basis Functions

We consider three representative types of basis functions in our framework: hat, Fourier, and Chebyshev, each exhibiting distinct approximation characteristics.

Hat basis functions are piecewise linear and locally supported, with each basis function being nonzero only over a small subinterval of the domain. This locality provides strong spatial interpretability and makes them well suited for capturing local variations in the source terms.

Fourier basis functions consist of globally supported trigonometric functions and are particularly effective for representing smooth and periodic inputs. Owing to their spectral approximation properties, high accuracy can often be achieved using only a small number of retained modes.

Chebyshev basis functions are orthogonal polynomials defined on a bounded interval and offer spectral accuracy for smooth, non-periodic functions. Compared to Fourier bases, Chebyshev bases avoid implicit periodicity assumptions and provide stable, high-order approximations near domain boundaries.

All three bases train stably and yield accurate operators, so the choice is a design decision rather than a restriction of the framework: on the periodic inputs of the main benchmarks the Fourier basis is the most accurate, whereas on non-periodic Matérn inputs the Chebyshev basis is (Table 5 and Appendix D.3).

B.4.2 Basis Function Configuration

This subsection specifies how each of the three bases is constructed and how its truncation order fixes the number of supervised training pairs.

Hat Basis Configuration. For a discretised domain with grid points x_k, $k = 0, 1, \ldots, N$, the hat function $\phi_k(x)$ is defined as:

$$\phi_k(x) = \begin{cases} \dfrac{x - x_{k-1}}{x_k - x_{k-1}}, & x \in [x_{k-1}, x_k], \\ \dfrac{x_{k+1} - x}{x_{k+1} - x_k}, & x \in [x_k, x_{k+1}], \\ 0, & \text{otherwise.} \end{cases} \tag{16}$$

Any piecewise linear function $f(x)$ can then be expressed as a linear combination of these basis functions:

$$f(x) \sim \sum_{k=1}^{N} c_k \phi_k(x), \qquad x \in [0, 1] \tag{17}$$

where the coefficients $\{c_k\}_{k=1}^{N}$ represent the expansion weights associated with the hat basis. Specifically, due to the interpolation property of hat functions, each coefficient c_k corresponds to the function value at the grid point x_k, i.e.,

$$c_k = f(x_k). \tag{18}$$

Figure 5 and Figure 6 present schematic illustrations of the hat basis functions and their corresponding reconstructions for the one-dimensional and two-dimensional Poisson equations, respectively.

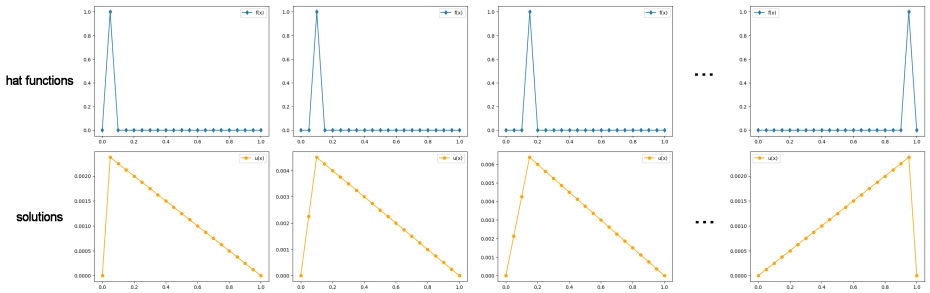

Figure 5: Illustration of one-dimensional hat basis functions and the corresponding solution of the Poisson equation.

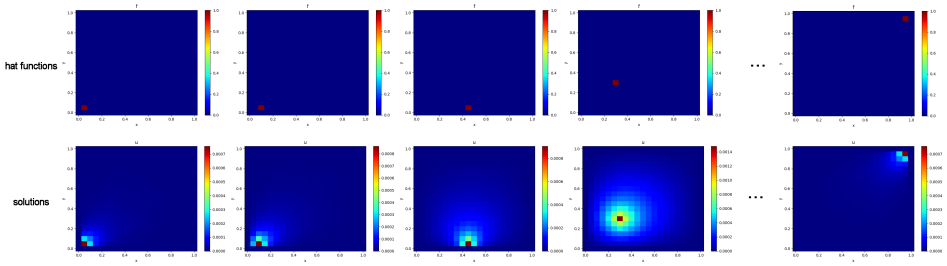

Figure 6: Illustration of two-dimensional hat basis functions and the corresponding solution of the Poisson equation.

Fourier Basis Configuration. For one-dimensional source terms, we use the standard Fourier series expansion:

$$f(x) \sim \frac{a_0}{2} + \sum_{n=1}^{\infty} \left[a_n \cos(2\pi n x) + b_n \sin(2\pi n x) \right]. \tag{19}$$

The corresponding coefficients can be computed using the following formula:

$$a_n = 2\int_0^1 f(x)\cos(2\pi nx)dx,$$
$$b_n = 2\int_0^1 f(x)\sin(2\pi nx)dx,$$

where $f(x)$ is defined on the interval $[0,1]$.

For two-dimensional source terms, we adopt a truncated Fourier basis of the form:

$$\begin{aligned} f(x,y) \sim a_{00} &+ \sum_{m=1}^{\infty}\left[a_{0m}\cos(2\pi my) + b_{0m}\sin(2\pi my)\right] \\ &+ \sum_{n=1}^{\infty}\left[a_{n0}\cos(2\pi nx) + c_{n0}\sin(2\pi nx)\right] \\ &+ \sum_{n=1}^{\infty}\sum_{m=1}^{\infty}\big[a_{nm}\cos(2\pi nx)\cos(2\pi my) + b_{nm}\cos(2\pi nx)\sin(2\pi my) \\ &\qquad + c_{nm}\sin(2\pi nx)\cos(2\pi my) + d_{nm}\sin(2\pi nx)\sin(2\pi my)\big]. \end{aligned} \tag{20}$$

The corresponding coefficients can be computed using the following formula:

$$\begin{aligned} a_{00} &= \int_0^1\int_0^1 f(x,y)\,dx\,dy, \\ a_{0m} &= 2\int_0^1\int_0^1 f(x,y)\cos(2\pi my)\,dx\,dy, \\ b_{0m} &= 2\int_0^1\int_0^1 f(x,y)\sin(2\pi my)\,dx\,dy, \\ a_{n0} &= 2\int_0^1\int_0^1 f(x,y)\cos(2\pi nx)\,dx\,dy, \\ c_{n0} &= 2\int_0^1\int_0^1 f(x,y)\sin(2\pi nx)\,dx\,dy, \\ a_{nm} &= 4\int_0^1\int_0^1 f(x,y)\cos(2\pi nx)\cos(2\pi my)\,dx\,dy, \\ b_{nm} &= 4\int_0^1\int_0^1 f(x,y)\cos(2\pi nx)\sin(2\pi my)\,dx\,dy, \\ c_{nm} &= 4\int_0^1\int_0^1 f(x,y)\sin(2\pi nx)\cos(2\pi my)\,dx\,dy, \\ d_{nm} &= 4\int_0^1\int_0^1 f(x,y)\sin(2\pi nx)\sin(2\pi my)\,dx\,dy. \end{aligned}$$

The function $f(x,y)$ is defined on the unit square domain $[0,1]\times[0,1]$.

In our experiments, we vary the truncation orders n and m of the Fourier expansion, which determine the number of retained basis functions and consequently the size of the supervised training dataset. While increasing n and m generally improves the approximation accuracy, it also leads to higher computational cost. Figures 7 and 8 illustrate representative examples of the Fourier basis functions and their corresponding PDE solutions for 1D and 2D Poisson equations, respectively.

Chebyshev Basis Configuration. In addition to Fourier and hat basis functions, we also employ Chebyshev basis functions, which are widely used in spectral methods due to their excellent approximation properties for smooth functions.

For one-dimensional source terms defined on $[0,1]$, we map the physical coordinate $x\in[0,1]$ to the standard Chebyshev domain $\xi\in[-1,1]$ via

$$\xi = 2x - 1.$$

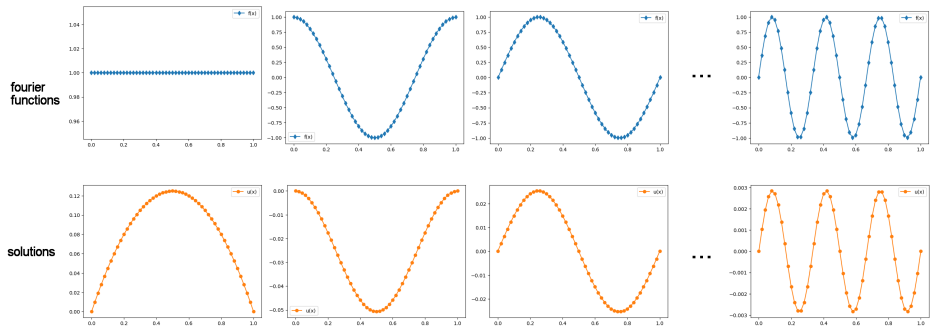

Figure 7: Illustration of one-dimensional Fourier basis functions and the corresponding solution of the Poisson equation.

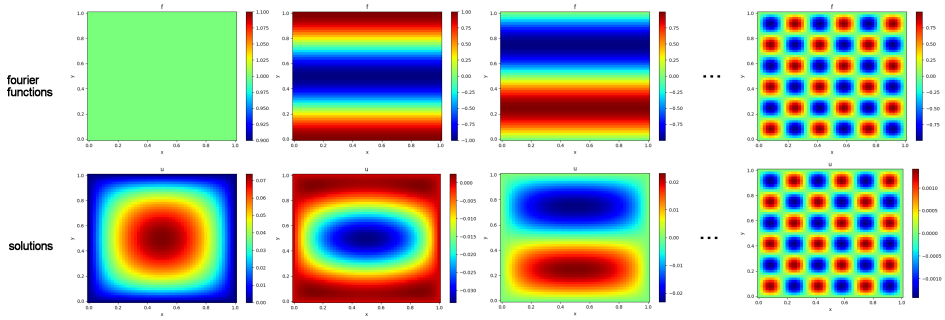

Figure 8: Illustration of two-dimensional Fourier basis functions and the corresponding solution of the Poisson equation.

The Chebyshev polynomials of the first kind are defined as

$$T_k(\xi) = \cos(k \arccos \xi), \qquad k = 0, 1, 2, \ldots$$

A function $f(x)$ can be approximated by a truncated Chebyshev expansion:

$$f(x) \sim \sum_{k=0}^{K} c_k T_k(2x - 1).$$

The expansion coefficients $\{c_k\}$ are given by the Chebyshev projection:

$$c_k = \frac{2}{\pi} c_k^* \int_{-1}^{1} \frac{f\left(\frac{\xi+1}{2}\right) T_k(\xi)}{\sqrt{1-\xi^2}}\, d\xi, \qquad k = 0, 1, \ldots, K,$$

where

$$c_k^* = \begin{cases} \frac{1}{2}, & k = 0, \\ 1, & k \geq 1, \end{cases}$$

so that the constant mode is weighted by $1/\pi$ and every higher mode by $2/\pi$. In practice, the integrals are evaluated numerically using Chebyshev–Gauss or Chebyshev–Gauss–Lobatto quadrature, which yields stable and accurate coefficient estimates.

Figure 9 and Figure 10 present schematic illustrations of the Chebyshev basis functions and their corresponding reconstructions for the one-dimensional and two-dimensional Poisson equations, respectively.

For two-dimensional source terms defined on $[0, 1]^2$, we use a tensor-product Chebyshev basis:

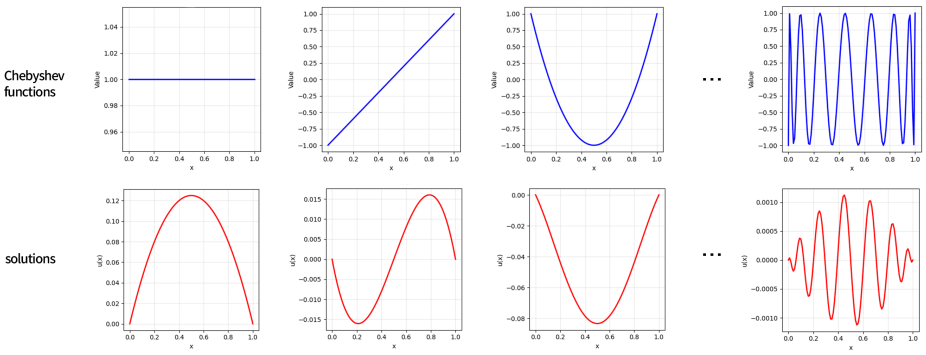

Figure 9: Illustration of one-dimensional Chebyshev basis functions and the corresponding solution of the Poisson equation.

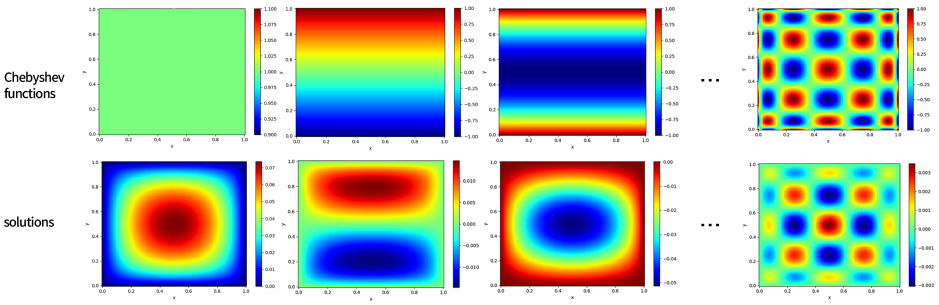

Figure 10: Illustration of two-dimensional Chebyshev basis functions and the corresponding solution of the Poisson equation.

$$f(x,y) \sim \sum_{i=0}^{K_x} \sum_{j=0}^{K_y} c_{ij} T_i(2x-1) T_j(2y-1).$$

The corresponding coefficients are computed in the same way,

$$c_{ij} = \frac{4}{\pi^2} c_i^* c_j^* \int_{-1}^{1} \int_{-1}^{1} \frac{f\left(\frac{\xi+1}{2}, \frac{\eta+1}{2}\right) T_i(\xi) T_j(\eta)}{\sqrt{1-\xi^2}\sqrt{1-\eta^2}} \, d\xi \, d\eta,$$

where c_i^* and c_j^* are defined as in the one-dimensional case. As with the Fourier basis, the truncation orders K_x and K_y control the number of retained basis functions and therefore the size of the supervised training dataset.

B.4.3 Vector-Valued Basis Expansion

For systems with multiple source terms (e.g., the Navier–Stokes equations), each component is expanded individually using basis functions and subsequently assembled into a vector-valued representation, as shown below:

$$\begin{pmatrix} f_1 \\ f_2 \end{pmatrix} \sim \alpha_1 \begin{pmatrix} \hat{h}_1 \\ 0 \end{pmatrix} + \alpha_2 \begin{pmatrix} \hat{h}_2 \\ 0 \end{pmatrix} + \cdots + \alpha_n \begin{pmatrix} \hat{h}_n \\ 0 \end{pmatrix} + \beta_1 \begin{pmatrix} 0 \\ \hat{g}_1 \end{pmatrix} + \beta_2 \begin{pmatrix} 0 \\ \hat{g}_2 \end{pmatrix} + \cdots + \beta_m \begin{pmatrix} 0 \\ \hat{g}_m \end{pmatrix},$$

where $\{\hat{h}_i\}_{i=1}^n$ and $\{\hat{g}_i\}_{i=1}^m$ are the basis sets used to construct the components f_1 and f_2 of the forcing term, respectively. The two counts need not be equal, and either set may consist of any of the bases introduced above.

These expansions offer interpretable priors on the input functions and are a key reason why the learned operator generalises from limited data. The coefficients follow directly from the basis in use:

for the Fourier basis through the standard Fourier integrals above, for the hat basis the coefficients are the function values at the grid points, and for the Chebyshev basis through the Chebyshev projection or a discrete Chebyshev transform.

C Complete Quantitative Results

This section reports the measurements behind the numbers quoted in the main text. It gives the full sweep over supervision settings that Table 2 summarizes, the spread of the error across individual test inputs behind the reported means, representative prediction fields, and the cost of the basis representation as the truncation order grows. All protocols follow Appendix B.3.

C.1 Full Benchmark Sweep

Table 8 reports the complete nonlinear benchmark results across all supervision settings, including the 5,000-sample configurations. Table 2 in the main text reports the entries that use 1,000 samples, so that the supervised and semi-supervised baselines are compared at a matched budget.

Table 8: Average relative L^2 error and training time (hours) of different baselines on nonlinear PDEs under different supervision paradigms. The results are evaluated on 100 unseen test samples with $\alpha_{\text{test}} = 3.5$, while the supervised training data are generated with $\alpha_{\text{train}} = 4$. Bold indicates the best result among the semi-supervised methods.

Eqn.	Baselines	Paradigm	# Sup.	# Unsup.	L^2 Error	Time
Poi.	DeepONet	Sup.	1,000	0	1.37e-2	0.370h
	DeepONet	Sup.	5,000	0	1.35e-2	1.855h
	Transolver	Sup.	1,000	0	2.11e-3	3.794h
	Transolver	Sup.	5,000	0	1.04e-3	17.452h
	PI-DeepONet	Semi-sup.	50	1,000	5.97e-2	4.963h
	PI-DeepONet	Semi-sup.	50	5,000	4.42e-2	20.127h
	PINO	Semi-sup.	50	1,000	4.27e-2	5.305h
	PINO	Semi-sup.	50	5,000	3.61e-2	23.911h
	PIBOs(ours)	Semi-sup.	50	1,000	**5.61e-3**	**0.369h**
RD.	DeepONet	Sup.	1,000	0	3.21e-2	0.385h
	DeepONet	Sup.	5,000	0	1.79e-2	1.822h
	Transolver	Sup.	1,000	0	2.71e-3	3.648h
	Transolver	Sup.	5,000	0	1.13e-3	17.690h
	PI-DeepONet	Semi-sup.	50	1,000	6.34e-2	5.214h
	PI-DeepONet	Semi-sup.	50	5,000	5.10e-2	22.817h
	PINO	Semi-sup.	50	1,000	5.88e-2	5.958h
	PINO	Semi-sup.	50	5,000	4.71e-2	25.117h
	PIBOs(ours)	Semi-sup.	50	1,000	**1.74e-2**	**0.523h**
NS.	DeepONet	Sup.	1,000	0	3.82e-2	0.403h
	DeepONet	Sup.	5,000	0	2.42e-2	1.917h
	Transolver	Sup.	1,000	0	5.10e-3	3.747h
	Transolver	Sup.	5,000	0	4.61e-3	16.238h
	PI-DeepONet	Semi-sup.	50	1,000	9.71e-2	8.053h
	PI-DeepONet	Semi-sup.	50	5,000	9.94e-2	32.446h
	PINO	Semi-sup.	50	1,000	8.91e-2	8.855h
	PINO	Semi-sup.	50	5,000	8.05e-2	33.236h
	PIBOs(ours)	Semi-sup.	50	1,000	**5.49e-2**	**0.490h**

C.2 Error Distributions across Test Samples

Accuracy and its spread. Table 9 reports the mean and standard deviation of the relative L^2 error of PIBOs on the three benchmarks. The linear models are trained on 50 supervised basis pairs,

and the nonlinear models on the same 50 pairs augmented with 1,000 unsupervised samples for the Picard-informed loss; both statistics are computed over the 100 unseen test inputs. The errors stay within a factor of 13 across the six settings, and the nonlinear problems, which receive no labeled solutions beyond the basis pairs, remain within a factor of three of their linear counterparts. The spread is widest for the nonlinear Navier–Stokes case ($5.49 \times 10^{-2} \pm 2.97 \times 10^{-2}$) and narrowest for the linear Poisson case ($4.40 \times 10^{-3} \pm 2.20 \times 10^{-3}$).

Table 9: The relative L^2 error for PIBOs on benchmark PDEs. We train on 50 supervised samples with $\alpha_{\text{train}} = 4$ and evaluate on 100 unseen test samples with $\alpha_{\text{test}} = 3.5$.

Equation	Type	Relative L^2 Error
Poi.	Linear	4.40e-3 $\pm$ 2.20e-3
	Nonlinear	5.61e-3 $\pm$ 2.40e-3
RD.	Linear	8.40e-3 $\pm$ 3.60e-3
	Nonlinear	1.74e-2 $\pm$ 9.81e-3
NS.	Linear	2.64e-2 $\pm$ 4.70e-3
	Nonlinear	5.49e-2 $\pm$ 2.97e-2

Spread across test inputs. Figure 11 shows the distribution behind each mean of Table 9, as a histogram, a cumulative distribution function, and a box plot for each of the six settings. The three views together separate the central tendency of the error from its dispersion and from the tails that the mean and standard deviation alone do not resolve.

C.3 Qualitative Prediction Fields

Figure 12 shows, for each benchmark, the input source term, the reference solution, the prediction of PIBOs, and the absolute error. The predicted fields reproduce the reference solutions, and the error maps remain small and spatially localized rather than spreading as a global bias. No spurious oscillations or boundary artifacts are visible in any of the settings shown.

C.4 Cost of the Basis Representation

While our PIBOs relies on the basis functions, the introduction of basis functions does not significantly increase the number of trainable parameters or the training time. We conducted relevant experimental analyses on the linear and nonlinear Poisson equations, with the detailed results presented in Table 10. Training time grows only mildly with the number of basis functions, and the peak GPU allocation is nearly unchanged, because the basis functions enter the manual expansion of the input functions rather than the network architecture. The linear and nonlinear models also have comparable parameter counts (approximately 122.24K each), so the improved accuracy is not obtained at the cost of substantially increased model complexity.

Table 10: Clock training time and GPU allocation for different numbers of basis functions.

	# basis functions	10	25	50	80	120	200
Linear Poisson	Training time	17.5s	42.0s	82.5s	140.4s	201.7s	293.4s
	GPU allocation (MB)	165.2	167.1	169.9	173.7	178.4	188.3
Nonlinear Poisson	Training time	1361s	1389s	1428s	1518s	1576s	1662s
	GPU allocation (MB)	284.8	286.5	289.3	293.1	297.9	311.3

C.5 Architecture-Agnostic Ablation

PIBOs is designed to be architecture-agnostic and to integrate with existing neural operators. Table 11 reports the relative L^2 error and the training time for four backbones under a fixed Fourier basis, with 50 supervised and 1,000 unsupervised samples. Accuracy and efficiency in the few-shot

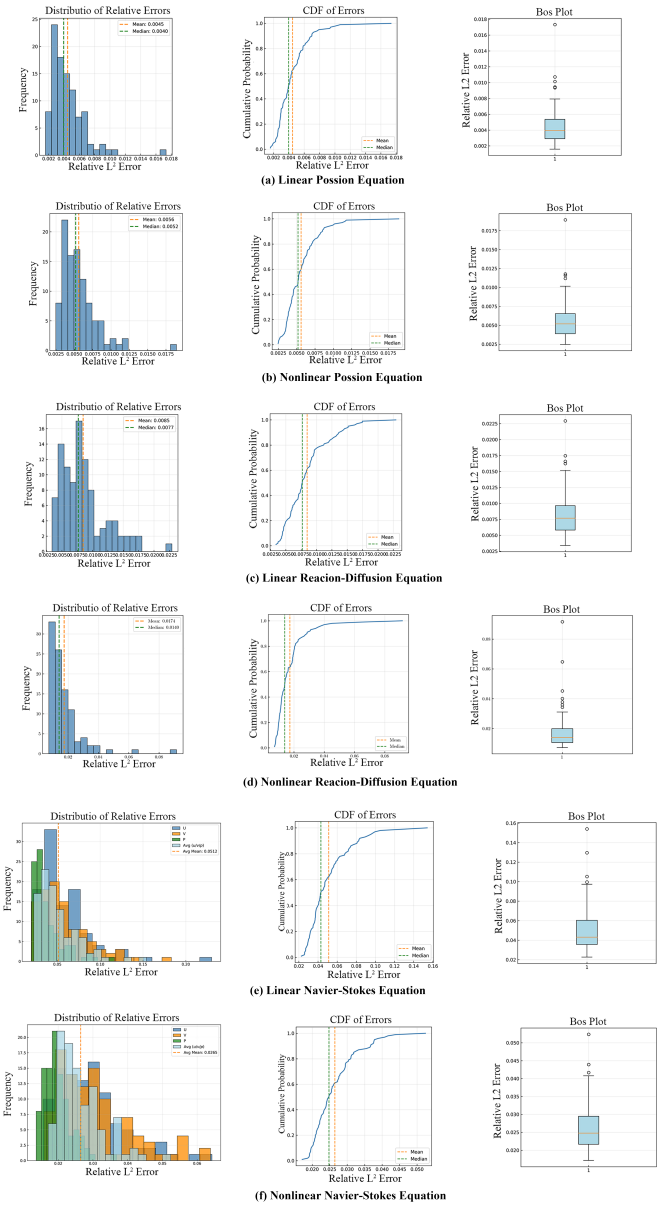

Figure 11: Statistical error distributions (histogram, CDF, and box plot) of relative L^2 errors on test datasets for linear and nonlinear Poisson, Reaction–Diffusion, and Navier–Stokes equations.

regime are consistent across backbones, which indicates that the framework transfers rather than depending on a particular operator architecture: the FNO backbone reaches a comparable nonlinear Poisson error at roughly five times the training time, the Transolver backbone is slower still, and the NIPS backbone degrades markedly because its attention kernel is less well matched to the pointwise nonlinearity of the benchmark.

D GENERALIZATION UNDER DISTRIBUTION SHIFT

The main text reports a single distribution shift between training and evaluation. This section takes that protocol apart and reports each component in isolation, so that the robustness of PIBOs can be attributed to a property of the construction rather than to the setting as a whole. Four shifts are tested:

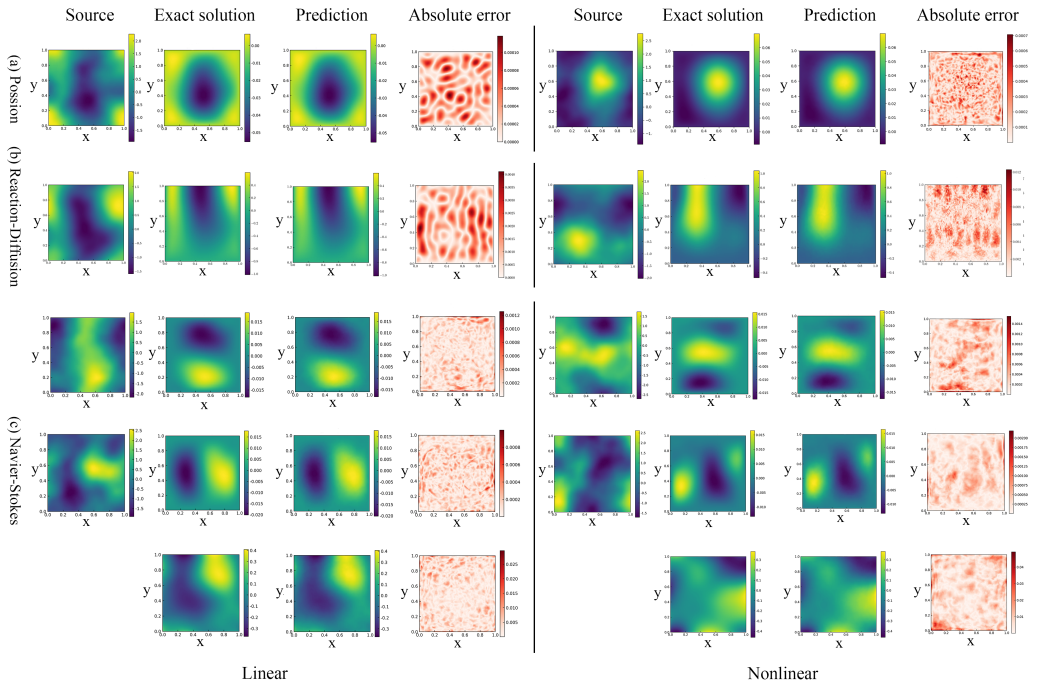

Figure 12: Prediction results of PIBOs on three benchmark PDEs. For each equation, the source, exact solution, model prediction, and absolute error are shown in the left, middle, and right panels, respectively.

Table 11: Ablation study on architecture-agnostic: L^2 Error / Training Time (hours).

Backbone	DeepONet	FNO	Transolver	NIPS
Linear Poi.	4.43e-3 / 0.022h	7.60e-3 / 0.098h	9.69e-3 / 0.245h	2.62e-2 / 0.272h
Nonlinear Poi.	5.61e-3 / 0.369h	5.21e-3 / 1.882h	8.48e-3 / 4.355h	3.61e-2 / 2.413h

the spectrum of the input changes while its mean is held fixed (Appendix D.1); the mean changes while the spectrum is held fixed (Appendix D.2); the inputs become non-periodic instead of periodic (Appendix D.3); and the inputs become rougher than anything seen during training (Appendix D.4). Unless stated otherwise, all protocols follow Appendix B.3: the metric is the relative L^2 error on 100 unseen test samples, PIBOs is trained with 50 supervised basis samples (augmented with 1,000 unsupervised samples in the nonlinear setting), and the supervised baselines are trained with 1,000 samples.

D.1 Spectral Shift

Test inputs are generated with decay parameters $\alpha_{\text{test}} \in \{2, 2.5, 3, 3.5, 4\}$ and no mean shift, while the models are trained at $\alpha_{\text{train}} = 4$. This experiment isolates the effect of a spectral distribution shift between training and testing inputs.

As shown in Table 12, all baseline methods exhibit a clear degradation in accuracy as the spectral smoothness of the test inputs deviates from the training distribution ($\alpha_{\text{train}} = 4$). In particular, DeepONet and Transolver suffer from substantial error increases under strong spectral shifts ($\alpha_{\text{test}} = 2$), while FNO shows improved robustness but still degrades noticeably as α decreases. NIPS demonstrates relatively stable performance across different spectral regimes, and achieves competitive accuracy under severe spectral shifts, benefiting from its Fourier-space kernelized attention design. However, its performance remains nearly constant across different values of α, indicating limited

adaptivity to changes in spectral smoothness. In contrast, PIBOs consistently achieves low errors across all tested values of α, and exhibits a clear improvement as the test distribution approaches the training regime.

Table 12: Mean relative L^2 error under spectral distribution shifts. All models are trained with forcing terms generated using $\alpha_{\text{train}} = 4$ and evaluated on test inputs with varying decay parameters $\alpha_{\text{test}} \in \{2, 2.5, 3, 3.5, 4\}$.

Method	$\alpha = 2$	$\alpha = 2.5$	$\alpha = 3$	$\alpha = 3.5$	$\alpha = 4$
DeepONet	6.07e-2	3.90e-2	2.67e-2	1.93e-2	1.51e-2
FNO	1.60e-2	7.57e-3	3.98e-3	2.40e-3	1.60e-3
Transolver	3.62e-2	1.47e-2	7.21e-3	4.11e-3	2.92e-3
NIPS	5.50e-3	5.40e-3	5.20e-3	5.03e-3	5.21e-3
PIBOs(ours)	**2.13e-3**	**1.70e-3**	**1.41e-3**	**1.34e-3**	**1.12e-3**

D.2 Mean Shift

With a concurrent spectral shift. The test forcing terms use $\alpha_{\text{test}} = 2$, so a spectral mismatch is present in every column, and a mean shift $f_{\text{test}} \leftarrow f_{\text{test}} + \delta$ with $\delta \in \{0.1, 0.2, 0.3, 0.4\}$ is added on top. This is the protocol of Section 4.3, reported there in Table 3 for the same four perturbation levels.

As shown in Table 3, most baseline methods degrade rapidly as the mean-shift magnitude increases under the spectral distribution shift. DeepONet and Transolver are particularly sensitive to global bias perturbations, with relative errors increasing sharply even for moderate values of δ. FNO is more robust than DeepONet but still accumulates error as the perturbation grows, and NIPS, although accurate under small mean shifts, deteriorates noticeably once δ is large.

In contrast, PIBOs achieves the lowest error at every perturbation level, and its error changes little as δ increases. The insensitivity is structural rather than learned: a constant offset lies in the span of the basis, so it changes only the lowest-order coefficient, which is computed exactly, whereas for the baselines that same offset falls outside the training distribution of their learned maps. Table 3 therefore compares the extrapolation behaviour of the learned maps at a fixed perturbation level; the spectral mismatch present in every column is the part that is genuinely out of distribution for PIBOs as well.

Without a spectral shift. The test decay parameter is fixed to $\alpha_{\text{test}} = 4$, matching the training distribution, and the same four mean-shift levels are applied. This removes the spectral mismatch and leaves the global bias as the only difference between training and test inputs.

As shown in Table 13, even in the absence of spectral mismatch, mean-shift perturbations alone can severely degrade the performance of most neural operator baselines. DeepONet, FNO, and Transolver exhibit rapidly increasing errors as δ grows, indicating high sensitivity to global bias changes in the forcing terms. PIBOs maintains consistently low relative errors across all perturbation levels, again because a constant offset is represented exactly by the basis; with the spectral shift removed, this table isolates the sensitivity of each learned map to an unseen input offset and nothing else.

D.3 Robustness to the Input Distribution: Matérn Random Fields

The main benchmarks draw source terms from periodic Gaussian random fields, which raises the question of whether the advantage of PIBOs reflects a genuine inductive bias or merely a fortunate alignment between periodic inputs and the default Fourier basis. To separate the two, we repeat the comparison on source terms sampled from Gaussian processes with a Matérn covariance, which produce non-periodic inputs whose smoothness is controlled by the Matérn parameter τ ($\tau = 0.5$ rough, $\tau = 1.5$ medium, $\tau = 2.5$ smooth). All methods are trained and evaluated on Matérn inputs, with 100 unseen test inputs. Here τ is unrelated to the viscosity ν of the Navier–Stokes equation.

Table 13: Mean relative L^2 error under mean-shift perturbations without spectral shift. All models are trained and evaluated on forcing terms generated with $\alpha_{\text{train}} = \alpha_{\text{test}} = 4$, while different mean-shift values δ are applied at test time.

Method	$\delta = 0.1$	$\delta = 0.2$	$\delta = 0.3$	$\delta = 0.4$
DeepONet	2.71e-1	5.27e-1	7.39e-1	8.97e-1
FNO	1.09e-1	2.19e-1	3.25e-1	4.12e-1
Transolver	2.23e-1	4.36e-1	6.18e-1	7.54e-1
NIPS	3.04e-2	2.04e-1	2.04e-1	2.04e-1
PIBOs(ours)	**1.03e-3**	**1.04e-3**	**9.37e-4**	**9.19e-4**

Two conclusions follow from Table 14. First, the advantage of PIBOs is not tied to a particular basis or input distribution: with a hat or Chebyshev basis and only 50 supervised samples, PIBOs already outperforms DeepONet trained on 1,000 samples at every smoothness level, a $20\times$ reduction in labeled data. At a matched budget of 1,000 samples, PIBOs with the Chebyshev basis attains the lowest error in all three columns, and the hat basis ranks second. Second, the ordering across bases follows the prediction of the framework rather than being incidental: Matérn inputs are non-periodic, so the Fourier basis, which serves as the default for the periodic source terms of the main text, is the weakest of the three, whereas the Chebyshev basis, which is suited to smooth non-periodic functions, is the strongest. This is the same qualitative conclusion as the basis study of Table 5, and it supports treating the basis as a design choice to be matched to the data rather than as a fixed component of the method.

Table 14: Relative L^2 error on source terms sampled from Matérn random fields with smoothness τ, evaluated on 100 unseen test inputs. PIBOs is shown with three basis choices at both the 50-sample few-shot budget and a matched 1,000-sample budget; all supervised baselines are trained on 1,000 samples.

Method	# Sup.	$\tau = 0.5$ (rough)	$\tau = 1.5$ (medium)	$\tau = 2.5$ (smooth)
DeepONet	1,000	9.73e-2	5.38e-2	3.92e-2
FNO	1,000	4.5e-2	1.6e-2	1.1e-2
Transolver	1,000	8.0e-2	3.2e-2	1.9e-2
NIPS	1,000	9.0e-3	7.1e-3	7.0e-3
DeepOMamba	1,000	3.0e-2	1.0e-2	8.0e-3
PIBOs (Fourier)	50	3.42e-1	8.91e-2	7.15e-2
PIBOs (hat)	50	3.08e-2	2.62e-2	2.23e-2
PIBOs (Chebyshev)	50	2.87e-2	1.14e-2	6.84e-3
PIBOs (Fourier)	1,000	8.5e-2	2.0e-2	1.4e-2
PIBOs (hat)	1,000	6.8e-3	5.2e-3	4.0e-3
PIBOs (Chebyshev)	1,000	**5.7e-3**	**2.0e-3**	**1.0e-3**

D.4 Rough-Input Stress Test

The spectral-shift study of Table 12 varies the decay parameter down to $\alpha_{\text{test}} = 2$. We additionally test a rougher regime, $\alpha_{\text{test}} = 1$, which lies well outside the range seen during training. Table 15 reports the result. PIBOs trained on 50 supervised samples attains 1.52×10^{-2}, roughly $16\times$ below DeepONet trained on 1,000 samples (2.41×10^{-1}) and $4.7\times$ below FNO on the same budget. Because the coefficients $\alpha_i(f)$ are computed analytically, the error of PIBOs under this shift remains governed by how well a finite Fourier basis resolves a rough input, rather than by the extrapolation behavior of a learned coefficient network, which is the failure mode observed in the baselines.

E Scope and Operating Range

Where the main text reports what the method achieves, this section reports where it stops. The nonlinear analysis of Appendix A holds under a contraction condition, and the first two subsections

Table 15: Rough-input stress test on the linear Poisson equation with source terms generated at $\alpha_{\text{test}} = 1$, which is rougher than any distribution used during training. PIBOs uses 50 supervised samples; DeepONet and FNO use 1,000.

Method	# Sup.	Relative L^2 error
DeepONet	1,000	2.41e-1
FNO	1,000	7.13e-2
PIBOs(ours)	50	**1.52e-2**

locate that condition in practice: the strength of the nonlinear term at which the advantage of PIBOs over a supervised baseline disappears (Appendix E.1), and the alternatives to the Picard scheme together with the reasons they were not adopted (Appendix E.2). The third subsection moves in the other direction and shows that the same construction carries over to an autoregressive task without labeled trajectories (Appendix E.3).

E.1 Nonlinearity Strength Sweep

The nonlinear analysis assumes that the Picard map is a contraction (Theorem 3.2), which confines the theory to regimes where the nonlinear correction is dominated by the linear operator. Table 16 measures where this assumption ceases to be useful in practice, by sweeping the coefficient λ of the nonlinear Poisson equation $-\Delta u + \lambda u^3 = f$ while holding the training protocol fixed.

PIBOs retains its advantage across the entire weakly and moderately nonlinear range, and the reference column $\lambda = 1$ reproduces the nonlinear Poisson result of Table 2. With 50 supervised samples (augmented by 1,000 unsupervised samples) it remains more accurate than DeepONet trained on 1,000 supervised samples up to $\lambda = 5$, a $50\times$ span in the strength of the nonlinear term, even though DeepONet uses $20\times$ more labeled data. The advantage narrows at $\lambda = 5$ (1.83×10^{-2} versus 1.94×10^{-2}) and reverses at $\lambda = 10$, where the contraction condition no longer holds and the Picard-consistent operator degrades faster than the purely supervised baseline. This is the practical boundary anticipated in Section 5: the method is not a general-purpose strongly nonlinear solver, and the sweep makes the boundary explicit rather than implicit.

Table 16: Nonlinearity strength sweep on the Poisson equation $-\Delta u + \lambda u^3 = f$. PIBOs uses 50 supervised samples augmented with 1,000 unsupervised samples; DeepONet is the supervised baseline trained on 1,000 samples. The column $\lambda = 1$ corresponds to the PIBOs entry for the nonlinear Poisson equation in Table 2.

Method	$\lambda = 0.1$	$\lambda = 0.5$	$\lambda = 1$	$\lambda = 2$	$\lambda = 5$	$\lambda = 10$
DeepONet (1,000 sup.)	1.32e-2	1.35e-2	1.37e-2	1.52e-2	1.94e-2	2.63e-2
PIBOs(ours)	**4.83e-3**	**5.24e-3**	**5.61e-3**	**7.92e-3**	**1.83e-2**	4.51e-2

E.2 Choice of the Iteration Scheme

The Picard reformulation $u = \mathcal{L}^{-1}(f - \mathcal{N}\mathcal{L}u)$ is one of several fixed-point schemes that can be used to construct an unsupervised loss, and Newton or quasi-Newton variants are the natural alternatives. PIBOs uses Picard because it is the only one of these schemes that can be evaluated without differentiating the nonlinear term: forming the Picard residual requires only $\mathcal{N}\mathcal{L}u$ itself, whereas a Newton step requires the Jacobian $\partial\mathcal{N}\mathcal{L}/\partial u$ and a quasi-Newton step an approximation to it. For the algebraic nonlinearities of the Poisson and reaction–diffusion benchmarks, $\mathcal{N}\mathcal{L}u$ is pointwise; for the convective term of Navier–Stokes it requires spatial derivatives, which we obtain by automatic differentiation of the neural representation, so the advantage over residual-based training is smaller in that case. A second consideration is that Picard iteration separates the linear inverse from the nonlinear correction, which is precisely the decomposition that allows the pretrained linear operator to serve both as the fixed-point map and as the initialization of the nonlinear model; Newton-type schemes mix the two and provide no comparable initialization. Whether Newton-type losses are

preferable in strongly nonlinear regimes, where the contraction condition of Theorem 3.2 fails, is left to future work.

E.3 An Autoregressive Extension

PIBOs can also be extended to autoregressive training for nonlinear PDEs that admit a clear linear approximation form (e.g., 1D diffusion–reaction, Burgers' equation, and incompressible Navier–Stokes). For example, we consider the nonlinear diffusion-reaction equation

$$u_t = \nu_1 u_{xx} + \nu_2(1-u)u, \qquad u(x,0) = u_0(x),$$

where the goal is to learn an operator $\mathcal{G} : u(x,t_n) \mapsto u(x,t_{n+1})$. We first approximate it with a linear PDE:

$$u_t = \nu_1 u_{xx} + c \cdot u, \qquad u(x,0) = u_0(x),$$

where c is an arbitrary constant. This linear PDE admits an explicit solution $u(x,t) = f(u_0, c)$ for an explicit function f, obtained via separation of variables under an appropriate basis. It admits the solution for the nonlinear PDE as $u(x,t) = f(u_0, \nu_2(1-u))$ implicitly if we set $c = \nu_2(1-u)$ formally. So we can construct the Picard-informed loss by

$$\mathcal{L} = \frac{1}{NTM} \sum_{i=1}^{N} \sum_{n=0}^{T} \sum_{j=1}^{M} \left| G_\theta(u_i(\cdot,t_n))(x_j) - f\left(u_0^i, \nu_2(1 - G_\theta(u_i))\right)(x_j,t_n) \right|^2 .$$

Table 17 shows that PIBOs achieves competitive accuracy and training time relative to supervised neural operators such as Transolver and DeepOMamba on the autoregressive task. The setting is more demanding for PIBOs: the baselines are trained on 1,000 labeled trajectories, whereas PIBOs requires no labeled data and relies only on the unsupervised Picard-informed loss.

Table 17: Relative L^2 error and training time on the autoregressive task.

Method	Supervision	L^2 Error	Training Time
DeepONet	1,000 labeled functions	2.17e-2	23min
Transolver	1,000 labeled functions	3.72e-3	87min
DeepOMamba	1,000 labeled functions	1.13e-3	23min
PIBOs(ours)	1,000 unlabeled functions	7.34e-3	29min

F Relation to Basis-Based Operator Learning

FEONet, SNO, SCLON, B2B, and PIBOs all reduce operator learning to a basis expansion, and they differ in one decisive respect: what is learned from data. This section states that difference exactly for the first four and identifies the property that separates PIBOs from all of them.

F.1 Conceptual Comparison

The differences between these methods are usefully separated along four dimensions: the basis that is used, the object that the network learns, the mechanism that accounts for nonlinearity, and the type of guarantee attached to the resulting approximation. The paragraphs below state, for each method, what its paper establishes along these dimensions.

FEONet (Lee et al., 2025) expands the solution in the classical finite-element nodal basis, which is tied to the mesh, and trains a network that maps an input function to the corresponding finite-element coefficients, supervising it with the FEM residual rather than with paired input–output solutions. Its convergence is analyzed through finite-element approximation theory, so the guarantee is inherited from the discretization and holds in distribution.

SNO (Fanaskov & Oseledets, 2023) and SCLON (Choi et al., 2024; Lee et al., 2026) work directly in the coefficient domain. SNO expands both the input and the output in Chebyshev or Fourier series and learns a mapping on the spectral coefficients themselves, which removes the aliasing bias

of grid-based operators. SCLON likewise expands in orthogonal functions (Fourier and Legendre series) but requires no paired data at all, relying on a loss derived from the weak formulation that satisfies the boundary conditions exactly. Neither method provides an a posteriori error bound for inputs outside the training distribution.

B2B (Ingebrand et al., 2025) learns a basis for each of the input and output spaces together with a possibly nonlinear map between their coefficients; the coefficients are computed by least squares, and for linear operators the coefficient map reduces to a single matrix transformation with a closed-form solution. Because these maps are fitted to paired input–output data, they must extrapolate from the training distribution to new inputs.

PIBOs differs from all of the above in that it learns no input-to-output map. For linear PDEs it learns only the N basis responses $G_\theta(\hat{h}_i)$ and recovers any new input by exact superposition with analytically computed coefficients $\alpha_i(f)$, so the out-of-distribution error is governed by the basis truncation error alone (Theorem 3.1). This is the essential distinction, and its empirical consequences are quantified in Section 4.4 and Appendix D.3. PIBOs is additionally evaluated under distribution shift (Tables 3, 12, and 13).