FIXED-POINT NEURAL OPERATOR: LEARNING TO ITERATIVELY CORRECT FOR PARAMETER-VARYING TRANSIENT PDE SOLVING

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural PDE solvers are typically direct predictors: they map problem data to a solution with a learned neural operator. As a result, they are agnostic to variations in PDE parameters and provide no mechanism for enforcing the governing physical constraints while solving. Classical iterative solvers address both issues by repeatedly evaluating the residual and using a preconditioner to map this feedback to a correction. Inspired by this *residual-to-correction* principle, we introduce the Fixed-point Neural Operator (FiNO), which uses a neural operator as a learned preconditioner. At each iteration, the PDE residual is evaluated on the current solution estimate and fed to the network, which predicts a correction that is applied repeatedly to refine the solution. This design incorporates equation feedback directly into inference and enables a shared solver to handle parameter-varying transient PDEs. Experiments across multiple problems show that the iterative solver substantially improves solution accuracy and physical consistency over direct neural operators, while providing stronger generalization across equation parameters.

1 INTRODUCTION

Traditional numerical methods treat solving as a process, not a prediction. Given an approximate solution, they evaluate its residual and use this measure of equation violation to construct a correction. Stationary methods such as preconditioned Richardson iteration (Richardson, 1911; Saad, 2003) repeatedly apply a fixed residual-to-correction rule, while Krylov methods such as CG (Hestenes et al., 1952) and GMRES (Saad & Schultz, 1986) construct corrections in subspaces generated from residual information. Preconditioning makes the connection to solution error explicit. For a linear system $Ax = b$, the residual and error satisfy $Ae = r$, and an inexpensive approximation to the inverse action of A transforms the residual into an approximate correction (Benzi, 2002). The common structure is an iterative loop that *evaluates equation violation and improves the current estimate*.

Most neural PDE solvers depart from this paradigm. Neural operators (Li et al., 2021; Lu et al., 2021; Kovachki et al., 2023), for instance, learn a direct mapping from initial or boundary conditions and problem parameters to the solution, typically evaluated in a single forward pass. While this approach facilitates rapid inference, it discards the intrinsic constraints imposed by the governing equations. Consequently, such models remain agnostic to variations in PDE parameters and cannot verify whether their predicted solutions satisfy the corresponding physical constraints.

Recent PDE foundation models have sought to mitigate these limitations through pre-training and fine-tuning strategies (Herde et al., 2024), or by encoding the PDEs via graph neural networks and equation-aware architectures (Ye et al., 2024; 2025; Zhu et al., 2025). Nevertheless, in these approaches, the models lack the capacity to directly perceive the PDE residual and dynamically adjust their behavior accordingly. Conversely, physics-informed learning, epitomized by Physics-Informed Neural Networks (PINNs) (Raissi et al., 2019), brings the governing equations directly into the optimization loop. However, by eschewing data-driven priors, PINNs suffer from optimization processes that are notoriously costly and prone to failure (Krishnapriyan et al., 2021).

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.

Existing approaches therefore put the learned prior and equation-based feedback to fundamentally distinct computational regimes, leaving open the critical challenge of unifying data-driven learning with residual-driven correction during inference.

Recent work has introduced iterative computation into neural PDE models, but the role of iteration differs substantially across these approaches. Generative approaches such as DiffusionPDE (Huang et al., 2024) and VideoPDE (Li et al., 2025) generate sequences of refined states, with physical information typically introduced as a conditioning signal rather than an explicitly evaluated equation residual. More recently, IRNO (Liu et al., 2026) wraps a pretrained neural operator with a learned refinement module and iteratively improves its prediction, primarily targeting the spectral bias of neural operators and the recovery of high-frequency components. Its refinement dynamics are driven by the learned prediction-refinement map rather than by solving the governing equation through an explicit residual feedback loop. HINTS (Zhang et al., 2024) is a numerical solver that explicitly uses the residual of a discretized linear PDE system to predict a correction. However, HINTS embeds the learned correction into a hybrid numerical solver and alternates it with hand-designed relaxation methods such as Jacobi or multigrid; the learned operator therefore serves as one component of an iterative solver for discretized linear systems.

We argue that neural PDE solvers should operate as iterative processes that update predictions directly from equation feedback, effectively learning a data-driven *residual-to-correction* map. Under this principle, fixed-point iteration provides a natural interface between a learned prior and the governing equation. We therefore introduce the **Fixed-point Neural Operator** (FiNO), which treats a neural operator as a learned preconditioner: the prescribed PDE residual is evaluated on the current iterate, and the network maps that residual, together with the current state, to an approximate correction. This process can be viewed as an extension of preconditioned Richardson iteration (Saad, 2003) to the space of nonlinear operators. We constructed three parameter-varying datasets based on the Navier-Stokes, Reaction-Diffusion, and Burgers equations to evaluate FiNO. Experimental results demonstrate the method's advantages in terms of solving precision and adherence to the governing PDEs, as well as its generalization capability.

The contributions of this work are as follows:

- We formulate neural PDE solving as learning a residual-conditioned correction operator for nonlinear transient trajectories, enabling a neural operator to serve as the learned correction mechanism of an iterative solving procedure.
- We propose the Fixed-point Neural Operator (FiNO), an iterative neural operator based on fixed-point iteration. Compared with direct predictors, FiNO explicitly conditions each correction on the evaluated PDE residual, providing awareness of equation adherence and physical consistency during inference.
- We construct parameter-varying benchmarks across multiple PDEs and demonstrate FiNO's advantages in solving precision, PDE adherence, and generalization.

2 METHODOLOGY

2.1 RESIDUAL-TO-CORRECTION FORMULATION FOR NONLINEAR PDE SOLVING

The preconditioned Richardson iteration (Richardson, 1911; Saad, 2003) is a foundational fixed-point scheme for solving linear systems of the form $Ax = b$. Given an initial guess $x^{(0)}$, the method generates a sequence of approximations via

$$x^{(k+1)} = x^{(k)} + \alpha_k M^{-1} r^{(k)}, \qquad r^{(k)} = b - Ax^{(k)}, \tag{1}$$

where $r^{(k)}$ denotes the residual at iteration k, α_k is a damping factor, and M is a preconditioner approximating A. The core mechanism is a *residual-to-correction* mapping: M^{-1} transforms the raw equation violation $r^{(k)}$ into an effective update direction, thereby accelerating convergence. The iteration terminates once the residual norm falls below a prescribed tolerance.

Although Eq. 1 is stated for finite-dimensional linear systems, the same residual-correction principle can be expressed without committing to a particular discretization. Let $\mathcal{X}$ and $\mathcal{Y}$ be Banach state and residual spaces, respectively, with essential initial and boundary data incorporated through a standard

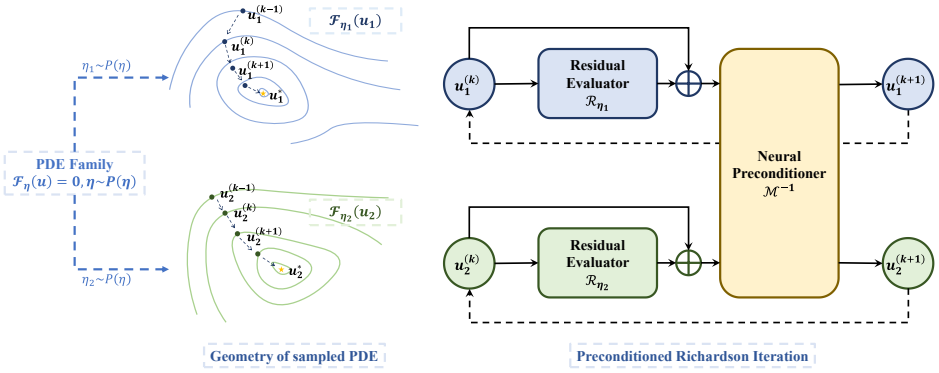

Figure 1: **Fixed-point Neural Operator (FiNO) architecture.** For each sampled PDE instance, the residual evaluator measures the equation violation of the current iterate. A shared neural preconditioner maps the current state and residual to a correction, and repeated updates drive the iterates toward the corresponding fixed-point solution.

lifting, and let

$$\mathcal{F}_\eta : \mathcal{X} \to \mathcal{Y}, \qquad \mathcal{F}_\eta(u) = 0, \qquad \mathcal{R}_\eta(u) = -\mathcal{F}_\eta(u), \tag{2}$$

where η collects equation parameters and $\mathcal{Y}$ is a residual space. The analogue of a preconditioner is a map from residual space back to the state space, and the corresponding abstract Richardson update is

$$u^{(k+1)} \;=\; u^{(k)} + \alpha_k\, \mathcal{M}_\eta^{-1} \circ \mathcal{R}_\eta\Big(u^{(k)}\Big)\,, \tag{3}$$

which provides the conceptual foundation for our proposed Fixed-point Neural Operator (FiNO), detailed in the following section.

2.2 Fixed-point Neural Operator

Building upon the operator formulation in Eq. 3, we now describe how to instantiate the preconditioning operator $\mathcal{M}_\eta^{-1}$ as a learnable neural module. Since it maps between function spaces, it is a natural candidate for parameterization via a neural operator. We refer to the resulting framework as the **Fixed-point Neural Operator** (FiNO).

Our solver decomposes the fixed-point update into two distinct operators. The first is the *PDE residual operator* $\mathcal{R}_\eta$, which evaluates the governing equation on the current iterate. Given the equation parameters η and the current solution estimate $u^{(k)}$, it computes

$$r^{(k)} = \mathcal{R}_\eta\Big(u^{(k)}\Big) = -\mathcal{F}_\eta\Big(u^{(k)}\Big)\,. \tag{4}$$

The second, as an instantiation of operator $\mathcal{M}_\eta^{-1}$, is the *neural preconditioner* $G_\theta : \mathcal{X} \times \mathcal{Y} \to \mathcal{X}$, a neural operator parameterized by θ that maps the pair $(u^{(k)},\, r^{(k)})$ to an approximate correction:

$$\delta^{(k)} = G_\theta\Big(u^{(k)},\, r^{(k)}\Big)\,. \tag{5}$$

The solution is then updated via the fixed-point iteration

$$u^{(k+1)} = u^{(k)} + \alpha^{(k)}\delta^{(k)}. \tag{6}$$

This decomposition preserves the iterative structure of classical solvers: at each step, the model explicitly evaluates the equation violation and proposes a correction informed by both the current state and the residual feedback.

Instead of employing a single end-to-end neural operator to map a current state and equation description directly to the next iterate (Ye et al., 2024; 2025; Zhu et al., 2025), we explicitly decouple equation evaluation from correction. This makes the iterative update depend on explicit, state-dependent equation feedback rather than only on an implicit learned representation of the equation.

In the linear Richardson setting, M^{-1} is a fixed approximation to the inverse of the discretized linear operator. FiNO generalizes the residual-to-correction principle in a different direction: the correction operator is learned as a state-dependent nonlinear map $G_\theta(u, r)$, where the residual $r = R_\eta(u)$ is recomputed from the governing equation at every iteration. Consequently, the learned operator provides a reusable correction procedure across a family of nonlinear transient PDE instances, with the equation residual supplying the instance-specific feedback.

2.3 Convergence Analysis

In this section, we give a conditional local convergence analysis of the FiNO iteration and use it to motivate the training objective. Under the assumptions stated below, the iteration converges locally to the target solution.

The FiNO map. Fix a damping factor $\alpha \in (0, 1]$. We write one step of the solver as the map

$$T_{\theta,\eta}(u) = u + \alpha\, G_\theta(u,\ \mathcal{R}_\eta(u))\,, \qquad G_\theta(u, 0) = 0, \tag{7}$$

defined on a neighborhood of a solution $u_\eta^\star$ satisfying $\mathcal{F}_\eta(u_\eta^\star) = 0$. The zero-residual consistency condition $G_\theta(u, 0) = 0$ ensures that the exact solution is a fixed point of $T_{\theta,\eta}$.

Linearization and preconditioning interpretation. To build intuition, consider the Fréchet derivative[1] of $T_{\theta,\eta}$ at $u_\eta^\star$. Because $u \mapsto G_\theta(u, 0)$ is identically zero, its derivative vanishes: $D_u G_\theta(u_\eta^\star, 0) = 0$. Moreover, $D\mathcal{R}_\eta(u_\eta^\star) = -J_\eta^\star$, where $J_\eta^\star = D\mathcal{F}_\eta(u_\eta^\star)$. The linearization of one FiNO step is therefore

$$DT_{\theta,\eta}(u_\eta^\star) = I - \alpha\, B_{\theta,\eta}^\star\, J_\eta^\star, \qquad B_{\theta,\eta}^\star := D_r G_\theta(u_\eta^\star, 0). \tag{8}$$

This is precisely the structure of a preconditioned Richardson step: the learned operator $B_{\theta,\eta}^\star$ acts as an approximate inverse of $J_\eta^\star$. The iteration is locally contractive whenever

$$\left\| I - \alpha\, B_{\theta,\eta}^\star\, J_\eta^\star \right\|_{\mathcal{X}\to\mathcal{X}} < 1, \tag{9}$$

which is the nonlinear analogue of the classical preconditioned Richardson condition. Unlike the linear setting, this does not require $\mathcal{F}_\eta$ itself to be linear.

Local convergence. The linearized picture extends to a rigorous local result via the contraction mapping principle.

Theorem 1 (Local contraction)**.** *Let $\mathcal{F}_\eta$ and G_θ be continuously Fréchet differentiable on a closed ball $B \subset \mathcal{X}$ centered at $u_\eta^\star$. If $T_{\theta,\eta}(B) \subseteq B$ and*

$$\sup_{u\in B} \left\| DT_{\theta,\eta}(u) \right\|_{\mathcal{X}\to\mathcal{X}} \le q < 1, \tag{10}$$

then $T_{\theta,\eta}$ has a unique fixed point in B, which is $u_\eta^\star$, and

$$\left\| u^{(k)} - u_\eta^\star \right\|_{\mathcal{X}} \le q^k \left\| u^{(0)} - u_\eta^\star \right\|_{\mathcal{X}}. \tag{11}$$

While Theorem 1 is clean, verifying Eq. 10 requires bounding the derivative of the full composed map. The following corollary gives a more directly trainable sufficient condition based on the accuracy of the learned correction.

Corollary 1 (Correction accuracy implies convergence)**.** *Suppose that, for some $\rho < 1$ and every $u \in B$,*

$$\left\| G_\theta(u, \mathcal{R}_\eta(u)) - \left(u_\eta^\star - u\right) \right\|_{\mathcal{X}} \le \rho\, \left\| u - u_\eta^\star \right\|_{\mathcal{X}}. \tag{12}$$

Then the damped FiNO update in Eq. 7 contracts toward $u_\eta^\star$ at rate $1 - \alpha(1 - \rho) < 1$.

Corollary 1 reveals that the training objective need not require the network to predict the full solution in one shot; it suffices for the network to approximate the *current error* $u_\eta^\star - u$ to within a relative accuracy $\rho < 1$, conditioned on the residual. This motivates our training loss design, which supervises the correction $\delta^{(k)}$ against the ground-truth error at each iteration (see Section 2.4).

[1] In finite-dimensional discretizations, the Fréchet derivative reduces to the usual Jacobian. We write $D_u G_\theta$ and $D_r G_\theta$ for derivatives with respect to the state and residual arguments, respectively.

Uniform convergence over a parameter family. For parameter-varying generalization, we require the contraction to hold uniformly over a family $\eta \in \mathcal{P}$. If the differentiability bounds are uniform and

$$\sup_{\eta \in \mathcal{P}} \left\| I - \alpha \, B^{\star}_{\theta,\eta} \, J^{\star}_{\eta} \right\| \leq q_0 < 1, \tag{13}$$

then there exists a uniform contraction factor $q < 1$ such that Eq. 11 holds simultaneously for every $\eta \in \mathcal{P}$ (see Appendix B). This clarifies the scope of parameter-varying generalization: the residual input exposes the instance-specific operator, while convergence across instances requires a shared neural preconditioner that is uniformly accurate on the chosen family.

Residual as a stopping criterion. Finally, we justify using residual reduction to monitor convergence. If $J^{\star}_{\eta}$ is boundedly invertible, then there exist a neighborhood B and constants $0 < c_\eta \leq C_\eta < \infty$ such that

$$c_\eta \left\| u - u^{\star}_{\eta} \right\|_{\mathcal{X}} \leq \left\| \mathcal{F}_\eta(u) \right\|_{\mathcal{Y}} \leq C_\eta \left\| u - u^{\star}_{\eta} \right\|_{\mathcal{X}}, \qquad u \in B. \tag{14}$$

Thus, locally, driving the residual to zero is equivalent to driving the solution error to zero. This requires local well-posedness and does not claim that an arbitrarily small residual identifies a unique solution globally. Proofs are given in Appendix B.

2.4 Training and Inference

Training objective. At each iteration k, the neural preconditioner outputs a correction $\delta^{(k)} = G_\theta(u^{(k)}, r^{(k)})$. We optimize a composite objective $\mathcal{L} = \mathcal{L}_{\text{sol}} + \lambda_{\text{cont}}\mathcal{L}_{\text{cont}} + \lambda_{\text{corr}}\mathcal{L}_{\text{corr}} + \lambda_{\text{zero}}\mathcal{L}_{\text{zero}}$, where each term serves a distinct purpose:

• **Solution Charbonnier loss ($\mathcal{L}_{\text{sol}}$):** We supervise the correction $\delta^{(k)}$ against the ground-truth trajectory $u^\star$ using a Charbonnier loss, which is more robust to sparse outliers than standard mean squared error:

$$\mathcal{L}_{\text{sol}} = \mathbb{E}\left[\sqrt{\|u^{(k)} + \delta^{(k)} - u^\star\|^2 + \epsilon^2} - \epsilon \right]. \tag{15}$$

• **Contraction loss ($\mathcal{L}_{\text{cont}}$):** To enforce the local contraction property established in Section 2.3, we explicitly penalize stagnation. We require both the solution error $e^{(k)} = \|u^{(k)} - u^\star\|$ and the residual norm $\|r^{(k)}\|$ to decrease by at least a target ratio $\rho < 1$ relative to the previous step:

$$\mathcal{L}_{\text{cont}} = \sum_{v \in \{e, \|r\|\}} \frac{\max\left(0, v^{(k)} - \rho v^{(k-1)}\right)}{v^{(k-1)} + \epsilon}. \tag{16}$$

• **Correction regularization ($\mathcal{L}_{\text{corr}}$):** To maintain training stability and prevent the network from generating disproportionately large updates, we penalize the magnitude of the correction, normalized by the preceding residual magnitude:

$$\mathcal{L}_{\text{corr}} = \mathbb{E}\left[\frac{\|\delta^{(k)}\|^2}{\|r^{(k)}\| + \epsilon} \right]. \tag{17}$$

• **Zero-residual consistency ($\mathcal{L}_{\text{zero}}$):** To satisfy the theoretical fixed-point requirement $G_\theta(u, 0) = 0$, we explicitly penalize non-zero corrections when the residual vanishes:

$$\mathcal{L}_{\text{zero}} = \mathbb{E}\left[\|G_\theta(u, 0)\|^2 \right]. \tag{18}$$

Unrolled optimization with replay buffer. Instead of training on independent single-step corrections, we unroll the fixed-point iteration for $K = 4$ steps and backpropagate gradients through the entire computational graph. This allows the network to learn corrections that are robust to its own accumulated errors. However, because deep convergence requires the network to handle states from deep iterations, training solely from initial guesses is insufficient. To address this, we maintain a *bucketed replay buffer* of depth 128 that stores intermediate states $(u^{(k)}, u^\star)$ categorized by their iteration depth k. This strategy ensures that the training distribution uniformly covers iteration depths, enabling the model to learn effective corrections across all stages of convergence.

Table 1: Performance evaluation. RMSE and residual RMS are channel-normalized for every method. Best and second-best results in each column are shown in bold and underlined, respectively.

		Navier–Stokes		Burgers		Reaction–Diffusion	
Method	Mode	RMSE	Residual RMS	RMSE	Residual RMS	RMSE	Residual RMS
DeepONet	Direct	0.6192	0.0693	0.6287	0.2731	0.3977	0.0615
	Residual	0.5430	0.1014	0.5262	0.1424	0.2585	0.0546
FNO	Direct	0.5730	0.0678	0.5690	0.1902	0.3293	0.0672
	Residual	0.5315	0.0929	0.2887	0.1208	0.2105	0.0654
PINO	Direct	0.6807	0.0507	0.5153	0.1349	0.3342	0.0650
	Residual	0.6634	<u>0.0459</u>	0.4461	0.1329	0.2160	0.0641
IRNO	Direct	0.5418	0.0576	0.5471	0.1846	0.2743	0.0563
	Residual	<u>0.4339</u>	0.0788	<u>0.2551</u>	<u>0.0903</u>	**0.2069**	<u>0.0410</u>
FiNO (Ours)		**0.1808**	**0.0239**	**0.2034**	**0.0881**	<u>0.2098</u>	**0.0197**

Residual-driven inference. At inference time, starting from an initial guess (e.g., copying the initial condition across the trajectory), we iteratively apply the damped update $u^{(k+1)} = u^{(k)} + \alpha G_\theta(u^{(k)}, r^{(k)})$. Since the exact solution is unknown at test time, we utilize the PDE residual as a proxy for convergence. This residual-driven stopping criterion prevents unnecessary computation and avoids over-iteration when the residual plateaus, ensuring efficient and reliable deployment.

3 Experiments

We conduct a comprehensive evaluation of FiNO on transient partial differential equations (PDEs) subject to varying parameters. Specifically, we assessed FiNO in terms of: (i) solution accuracy across multiple scenarios, (ii) generalization capability regarding PDE parameter variations, (iii) stability during long-term rollouts, and (iv) the effectiveness of its individual components.

3.1 Experimental setup

PDEs and data. We evaluate all methods on three parametric PDE benchmarks with periodic boundary conditions: **2D Navier–Stokes** (NS) flow, a **2D Reaction–Diffusion** (RD) system, and the **1D viscous Burgers** equation (Li et al., 2021; LeVeque, 2007). The NS data contain vorticity fields on a 256×256 grid with $\Delta t = 0.01$, $\mathrm{Re} \sim \mathrm{LogUniform}(10^2, 10^4)$ and a randomized Kolmogorov forcing. The RD data contain two-channel concentration fields on a 128×128 grid with $\Delta t = 4$, while all diffusion and reaction coefficients vary across trajectories. The Burgers data is generated on a 256-point grid with $\Delta t = 0.02$, randomized viscosity, forcing, and initial conditions. All reported operator-learning results use $T = 32$ snapshots (one initial state predicting 31 target states). More details are provided in Appendix D.

Baselines. We compare against four representative neural-operator baselines: **FNO** (Li et al., 2021), **DeepONet** (Lu et al., 2021), **PINO** (Li et al., 2024), and the **IRNO** (Liu et al., 2026). Each method receives the initial field and predicts the following 31 states of the $T = 32$ window; PINO uses the same FNO backbone with an additional PDE-residual loss, whereas IRNO iteratively refines a pretrained FNO prediction. To distinguish the benefit of our fixed-point formulation from the simpler benefit of exposing PDE information, we evaluate every baseline in both a direct form and with a common residual extension. More details are provided in Appendix E.

3.2 Main results and iterative behavior

Quantitative comparison. Table 1 compares solution accuracy and equation satisfaction on held-out $T = 32$ windows. Detailed equation satisfaction metric is provided in Appendix C.3. FiNO obtains the lowest normalized RMSE on Navier–Stokes and Burgers and remains close to the best result on Reaction–Diffusion. It also achieves the lowest residual RMS across all three equations.

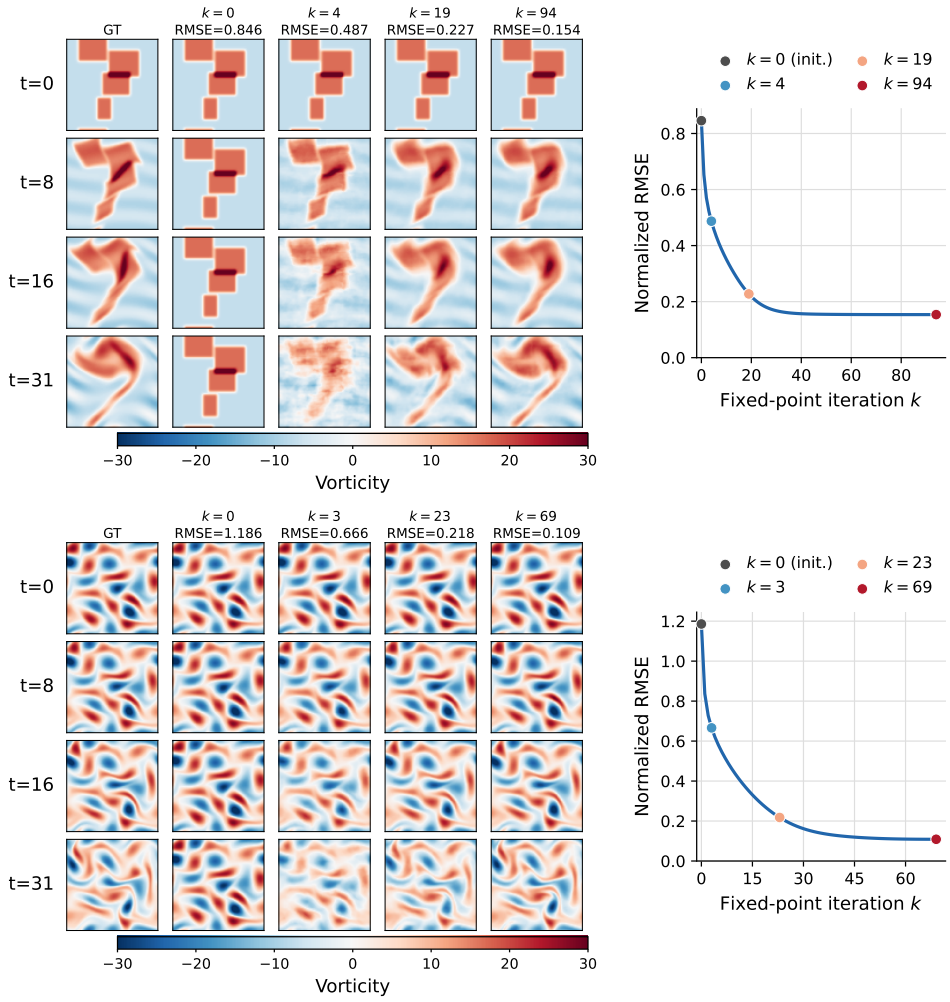

Figure 2: **Evolution of Navier–Stokes solutions across fixed-point iterations.** Each example shows the ground truth, selected iterates, and normalized RMSE as a function of iteration. Starting from a constant-in-time initialization, repeated residual-conditioned corrections progressively recover the trajectory at $t \in \{8, 16, 31\}$. The displayed errors decrease substantially in both representative cases, with the converged iterates recovering the dominant structures and fine-scale variations.

Supplying residual features to a direct predictor often improves its solution error, but does not consistently reduce its final PDE residual. The larger and more consistent gains of FiNO therefore come from iteration, rather than from residual features alone. Qualitative comparisons are provided in Appendix F.1.

As shown in Figure 2, early iterations recover the large-scale vorticity layout, while later corrections sharpen smaller structures and reduce the error throughout the temporal window. The two examples require different numbers of iterations, which motivates an instance-adaptive stopping rule instead of a fixed iteration count.

Residual-based iterate selection. Figure 3 separates the output selected by the minimum residual from the last iterate permitted by a given computation budget. Increasing the budget initially improves both prediction error and physics consistency, with most gains realized during the initial refinement stage. Beyond this point, the current iterate offers negligible accuracy improvement while its residual eventually increases. Selecting the lowest-residual iterate stabilizes both curves.

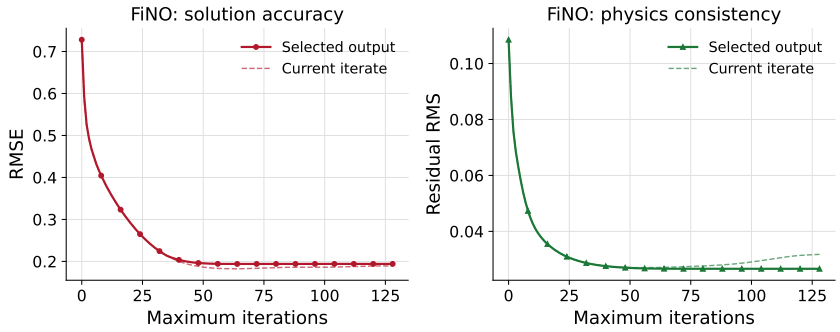

Figure 3: **Accuracy and physics consistency versus the available iteration budget on Navier–Stokes.** Dashed curves evaluate the current iterate at the maximum allowed iteration, whereas solid curves evaluate the iterate selected by the lowest observed residual. Both solution RMSE and residual RMS fall rapidly during the early iterations. Once the residual plateaus and the latest iterate begins to over-iterate, residual- based selection preserves a low-defect solution without access to the ground truth.

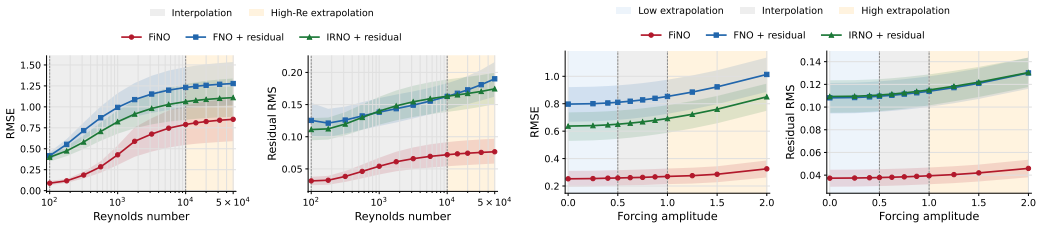

Figure 4: **Navier–Stokes generalization across Reynolds number and forcing amplitude.** Left: Reynolds-number sweep with forcing amplitude fixed at $A = 0.75$; training covers $\mathrm{Re} \in [10^2, 10^4]$. Right: forcing amplitude sweep at $\mathrm{Re} = 1000$; training covers $A \in [0.5, 1.0]$. Curves show trajectory-level means and shaded regions show 95% confidence intervals. Gray regions denote interpolation and colored regions denote extrapolation. Lower is better.

3.3 Generalization across equation parameters

The explicit residual interface also improves generalization when the operator parameters move away from the training distribution. In the Reynolds-number sweep in Figure 4, FiNO has the lowest solution error and residual RMS throughout the tested parameter axis. The advantage is especially clear in the interpolation regime and remains pronounced in the high-Reynolds extrapolation region, where all methods degrade but FiNO retains the lowest error and residual. The absence of a sharp transition at the training boundary further indicates smooth, rather than memorized, parameter adaptation.

We separately vary the Kolmogorov forcing amplitude from $A = 0$ to $A = 2$ while training only on $A \in [0.5, 1.0]$. At the strongest tested forcing, FiNO remains clearly more accurate than both residual-augmented baselines, and its residual also remains substantially lower throughout the sweep. The gradual degradation outside both training intervals indicates that residual feedback provides a useful adaptation signal, although it does not eliminate the difficulty of increasingly turbulent or strongly forced dynamics.

3.4 Long-horizon rollout

Although all models are trained only on $T = 32$ windows, their temporal reach can be extended by repeatedly shifting the window forward. We use a stride of 8, and initialize next window with current predictions. Figure 5 shows that FNO accumulates error rapidly after the training horizon, whereas IRNO becomes unstable late in the rollout. FiNO degrades much more gradually and remains stable through the final frame. Its cumulative solution error and residual remain far below those

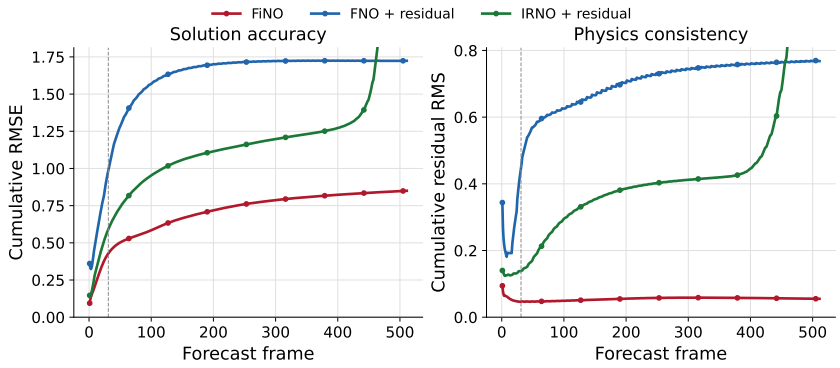

Figure 5: **Navier–Stokes rollout from $T = 32$ training windows to a 512-frame horizon.** Each method is applied in overlapping $T = 32$ windows; the window advances by 8 steps at a time. Curves report cumulative normalized RMSE (left) and cumulative residual RMS (right) over 100 test trajectories. The vertical dashed line marks the training horizon.

Table 2: **Loss ablation on Navier–Stokes ($T = 32$).** Each row removes the indicated auxiliary term from the full FiNO objective. Lower is better for both errors; mean iterations are reported descriptively because premature residual stagnation can also trigger early stopping.

Loss configuration	RMSE ↓	Res. RMS ↓	Mean iter.
Full	**0.1808**	0.0239	44.0
Without correction regularization	0.1838	0.0240	43.7
Without zero-residual consistency	0.1866	0.0241	43.3
Without solution contraction	0.2044	**0.0232**	40.1
Without residual contraction	0.3722	0.0549	21.6
Solution only	0.4589	0.0639	17.4

of the baselines throughout the horizon. Thus, reevaluating the PDE residual inside each window substantially limits compounding error, even though the model has never been trained on a 512-frame trajectory.

3.5 Loss ablation

Table 2 isolates the contribution of the training losses. Removing either correction regularization or zero-residual consistency causes a modest increase in solution error. The contraction terms are more consequential: removing either contraction signal produces a much larger accuracy loss, especially when residual contraction is omitted. Training with the solution loss alone performs worst on both solution accuracy and residual consistency. The low mean iteration counts of these weaker variants indicate earlier stagnation at a poorer solution. The full objective provides the best solution accuracy while retaining nearly the best residual consistency, supporting the joint use of state-space and residual-space contraction supervision.

4 Conclusion

We introduced FiNO, a fixed-point neural operator that treats the PDE residual as an inference-time correction signal. Across Navier–Stokes, Reaction–Diffusion, and Burgers benchmarks, residual-conditioned iteration improves solution accuracy and equation satisfaction, generalizes across PDE parameters, and remains stable in long-horizon rollouts. These results support learning neural operators as reusable correction procedures rather than one-shot predictors. Future work will extend this perspective to a broader range of PDE families, more varied geometries, and nonuniform or unstructured meshes, while developing discretization-agnostic residual interfaces with corresponding fixed-point analyses.

5 AI USE STATEMENT

In this work, we used generative AI tools for generate synthetic data sets, assist in the writing of proofs, implement methods, assist with translation, clean and reformat dataset, support qualitative and thematic data analysis. We have not used generative AI tools for help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, propose or refine hypotheses, design or provide feedback on research methodology or experiments, interpret results. Additionally, we used generative AI tools for summarize or analyse existing literature, edit a research paper to improve readability, identify relevant literature. We have reviewed all AI-assisted work. We visualized and programmatically verified the authenticity and physical consistency of AI-generated data. We reviewed the proofs and translations generated with AI assistance. We also cross-verified the authenticity and completeness of the literature identified by the AI against our own database. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Michele Benzi. Preconditioning techniques for large linear systems: a survey. *Journal of computational Physics*, 182(2):418–477, 2002.

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

Achi Brandt. Multi-level adaptive solutions to boundary-value problems. *Mathematics of Computation*, 31(138):333–390, 1977. doi: 10.1090/S0025-5718-1977-0431719-X.

Susanne C. Brenner and L. Ridgway Scott. *The Mathematical Theory of Finite Element Methods*. Springer, 3 edition, 2008. doi: 10.1007/978-0-387-75934-0.

Donglin Chen, Xiang Gao, Chuanfu Xu, Siqi Wang, Shizhao Chen, Jianbin Fang, and Zheng Wang. Flowdnn: a physics-informed deep neural network for fast and accurate flow prediction. *Frontiers of Information Technology & Electronic Engineering*, 23(2):207–219, 2022a.

Yuyan Chen, Bin Dong, and Jinchao Xu. Meta-MgNet: Meta multigrid networks for solving parameterized partial differential equations. *Journal of Computational Physics*, 455:110996, 2022b. doi: 10.1016/j.jcp.2022.110996.

Weinan E and Bing Yu. The deep ritz method: A deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018. doi: 10.1007/s40304-018-0127-z.

Peter Gray and Stephen K. Scott. Autocatalytic reactions in the isothermal, continuous stirred tank reactor. *Chemical Engineering Science*, 39(6):1087–1097, 1984.

Daniel Greenfeld, Meirav Galun, Ronen Basri, Irad Yavneh, and Ron Kimmel. Learning to optimize multigrid PDE solvers. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2415–2423. PMLR, 2019. URL https://proceedings.mlr.press/v97/greenfeld19a.html.

Maximilian Herde, Bogdan Raonić, Tobias Rohner, Roger Käppeli, Roberto Molinaro, Emmanuel De Bezenac, and Siddhartha Mishra. Poseidon: Efficient foundation models for pdes. *Advances in Neural Information Processing Systems*, 37:72525–72624, 2024.

Magnus R Hestenes, Eduard Stiefel, et al. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49(6):409–436, 1952.

Jun-Ting Hsieh, Shengjia Zhao, Stephan Eismann, Lucia Mirabella, and Stefano Ermon. Learning neural PDE solvers with convergence guarantees. *arXiv preprint arXiv:1906.01200*, 2019. doi: 10.48550/arXiv.1906.01200.

Wenjie Hu, Sidun Liu, Peng Qiao, Zhenglun Sun, and Yong Dou. Transolver is a linear transformer: Revisiting physics-attention through the lens of linear attention. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 408–416, 2026.

Jiahe Huang, Guandao Yang, Zichen Wang, and Jeong Joon Park. Diffusionpde: Generative pde-solving under partial observation. *Advances in Neural Information Processing Systems*, 37: 130291–130323, 2024.

D. A. Knoll and D. E. Keyes. Jacobian-free Newton–Krylov methods: A survey of approaches and applications. *Journal of Computational Physics*, 193(2):357–397, 2004. doi: 10.1016/j.jcp.2003.08.010.

Alena Kopaničáková and George Em Karniadakis. DeepONet-based preconditioning strategies for solving parametric linear systems of equations. *SIAM Journal on Scientific Computing*, 47(1): A151–A181, 2025. doi: 10.1137/24M162861X.

Alena Kopaničáková, Youngkyu Lee, and George Em Karniadakis. Leveraging operator learning to accelerate convergence of the preconditioned conjugate gradient method. *Machine Learning for Computational Science and Engineering*, 1:39, 2025. doi: 10.1007/s44379-025-00039-7.

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.

Aditi Krishnapriyan, Amir Gholami, Shandian Zhe, Robert Kirby, and Michael Mahoney. Characterizing possible failure modes in physics-informed neural networks. *Advances in neural information processing systems*, 34:26548–26560, 2021.

Randall J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge University Press, 2002. doi: 10.1017/CBO9780511791253.

Randall J. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations*. Society for Industrial and Applied Mathematics, 2007. doi: 10.1137/1.9780898717839.

Edward Li, Zichen Wang, Jiahe Huang, and Jeong Joon Park. Videopde: Unified generative pde solving via video inpainting diffusion models. *arXiv preprint arXiv:2506.13754*, 2025.

Zongyi Li, Nikola 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/IMS Journal of Data Science*, 1(3):1–27, 2024.

Phillip Lippe, Bas Veeling, Paris Perdikaris, Richard Turner, and Johannes Brandstetter. Pde-refiner: Achieving accurate long rollouts with neural pde solvers. *Advances in Neural Information Processing Systems*, 36:67398–67433, 2023.

Xiaotian Liu, Shuyuan Shang, Xiaopeng Wang, Pu Ren, and Yaoqing Yang. Iterative refinement neural operators are learned fixed-point solvers: A principled approach to spectral bias mitigation. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=4lAS8tI23u.

Yan Liu, Qingyang Zhang, Xinhai Chen, Chuanfu Xu, Qinglin Wang, and Jie Liu. Lkflownet: A deep neural network based on large kernel convolution for fast and accurate nonlinear fluid-changing prediction. *Physics of Fluids*, 36(9), 2024.

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.

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.

Lewis Fry Richardson. The approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam. *Philosophical Transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character*, 210:307–357, 1911.

Youcef Saad and Martin H Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing*, 7(3):856–869, 1986.

Yousef Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, 2 edition, 2003. doi: 10.1137/1.9780898718003.

Justin Sirignano and Konstantinos Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, 2018. doi: 10.1016/j.jcp.2018.08.029.

Lloyd N. Trefethen. *Spectral Methods in MATLAB*. Society for Industrial and Applied Mathematics, 2000. doi: 10.1137/1.9780898719598.

Kiwon Um, Robert Brand, Yun Fei, Philipp Holl, and Nils Thuerey. Solver-in-the-loop: Learning from differentiable physics to interact with iterative PDE-solvers. In *Advances in Neural Information Processing Systems*, volume 33, pp. 6111–6122, 2020.

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. doi: 10.1126/sciadv.abi8605.

Haixu Wu, Huakun Luo, Haowen Wang, Jianmin Wang, and Mingsheng Long. Transolver: A fast transformer solver for pdes on general geometries. *arXiv preprint arXiv:2402.02366*, 2024.

Zhanhong Ye, Xiang Huang, Leheng Chen, Hongsheng Liu, Zidong Wang, and Bin Dong. PDEformer: Towards a foundation model for one-dimensional partial differential equations. In *ICLR 2024 Workshop on AI4DifferentialEquations In Science*, 2024.

Zhanhong Ye, Zining Liu, Bingyang Wu, Hongjie Jiang, Leheng Chen, Minyan Zhang, Xiang Huang, Qinghe Meng Zou, Hongsheng Liu, Bin Dong, et al. Pdeformer-2: A versatile foundation model for two-dimensional partial differential equations. *arXiv preprint arXiv:2507.15409*, 2025.

Enrui Zhang, Adar Kahana, Alena Kopaničáková, Eli Turkel, Rishikesh Ranade, Jay Pathak, and George Em Karniadakis. Blending neural operators and relaxation methods in PDE numerical solvers. *Nature Machine Intelligence*, 6(11):1303–1313, 2024. doi: 10.1038/s42256-024-00910-x.

Qian-Ze Zhu, Paul Raccuglia, and Michael P Brenner. Generalizing pde emulation with equation-aware neural operators. *arXiv preprint arXiv:2511.09729*, 2025.

A Related Work

Numerical PDE solvers. Classical solvers first discretize the governing equations using finite differences (LeVeque, 2007), finite volumes (LeVeque, 2002), finite elements (Brenner & Scott, 2008), or spectral methods (Trefethen, 2000). Nonlinear systems are commonly linearized through Newton–Krylov methods (Knoll & Keyes, 2004), after which the resulting sparse systems are solved iteratively. Stationary schemes such as Jacobi and Gauss–Seidel (Saad, 2003) and Richardson iteration (Richardson, 1911) repeatedly update an approximate solution, whereas Krylov methods such as CG (Hestenes et al., 1952) and GMRES (Saad & Schultz, 1986) construct increasingly expressive search spaces from residual information. Multigrid (Brandt, 1977) further couples corrections across resolutions to remove errors at complementary spatial frequencies. Across these families, preconditioning is often decisive: an approximate inverse maps the current residual toward the unknown error, improves the effective conditioning, and accelerates convergence (Benzi, 2002). Effective preconditioners, however, typically encode substantial knowledge of the operator, discretization, geometry, or coefficient regime. Thus, classical solvers use equation feedback exactly and transparently, but do not by themselves learn a reusable prior from a distribution of solved problems.

Direct neural PDE solvers. Neural PDE solvers complement numerical methods by learning such priors from data. Typically, neural operators (Lu et al., 2021; Li et al., 2021; Brandstetter et al., 2022; Liu et al., 2024; Kovachki et al., 2023; Wu et al., 2024; Hu et al., 2026) amortize the cost of solving PDE instances into training and then predict a new solution with one or a small number of network evaluations. This yields fast inference, but the usual predictor has no state-dependent mechanism for checking its output against the governing equation. Dependence on PDE parameters and boundary conditions must therefore be captured implicitly from the training distribution, and physical consistency is not automatically recovered when those quantities shift. Recent foundation models broaden the learned prior through large-scale pretraining, as in Poseidon (Herde et al., 2024), or through explicit equation representations, as in PDEformer (Ye et al., 2024; 2025), and equation-aware neural operators (Zhu et al., 2025). Encoding target equation improves cross-equation transfer, but these models remain primarily direct predictors, lacking a substantive understanding of governing equations and their feedback.

Physics-informed learning. A complementary line of work introduces the governing equation into the learning objective. The Deep Ritz method (E & Yu, 2018) minimizes a variational energy, while DGM (Sirignano & Spiliopoulos, 2018) and PINNs (Raissi et al., 2019) optimize coordinate networks using differential equation residuals and boundary conditions. Such instance-wise optimization does not amortize a strong solution prior in the same manner as supervised operator learning and can be difficult to optimize, particularly for multiscale or long-time dynamics (Krishnapriyan et al., 2021). Physics-informed DeepONet (Wang et al., 2021; Chen et al., 2022a) and PINO (Li et al., 2024) instead regularize operator learning with residual losses, combining data and physics during training. This generally improves physical consistency and can reduce the required labeled data, but the trained operator is still normally applied as a direct predictor at test time.

Learned iterative solvers and preconditioners. Several works move learned computation closer to classical iterative solving. Solver-in-the-Loop (Um et al., 2020) trains correction models through a differentiable numerical solver so that they remain stable under their own recurrent inputs. Hsieh et al. (2019) learn a linear correction to a valid Jacobi update while preserving its fixed point and convergence guarantees. Learned multigrid methods instead target particular solver components: Greenfeld et al. (2019) predict prolongation operators, and Meta-MgNet (Chen et al., 2022b) learns multigrid structure across parameterized PDEs. More recent neural preconditioners use DeepONet to construct preconditioners for parametric linear systems (Kopaničáková & Karniadakis, 2025) or deflation spaces for PCG (Kopaničáková et al., 2025). HINTS (Zhang et al., 2024) feeds the residual of a discretized linear differential equation to DeepONet to predict a correction. Its learned correction is, however, integrated with hand-designed relaxation methods such as Jacobi, Gauss-Seidel, and multigrid, exploiting their complementary behavior across frequency modes. These studies establish that learned components can substantially improve a numerical iteration, but they primarily address stationary linear systems or a specific component of a fixed classical solver.

The proposed FiNO is related to this residual-driven perspective but differs in both the problem formulation and the role of the learned operator. We consider nonlinear transient PDEs with varying

equation parameters and evaluate the governing-equation residual on the current solution trajectory at every iteration. The neural operator directly maps the current state and its equation residual to the next correction, so the learned residual-to-correction map defines the iterative correction dynamics rather than augmenting a predefined relaxation scheme. This formulation allows equation feedback to determine the update throughout the solving process and provides a natural basis for analyzing the learned iteration as a nonlinear fixed-point map.

In parallel, PDE-Refiner (Lippe et al., 2023), DiffusionPDE (Huang et al., 2024), and VideoPDE (Li et al., 2025) perform multi-step generative refinement, while IRNO (Liu et al., 2026) repeatedly refines a pretrained neural-operator prediction. Their iterations are learned denoising or refinement processes rather than a general residual-to-correction interface.

B Proofs for the Operator Analysis

Proof of Theorem 1. For any $u, v \in B$, the mean-value inequality in Banach spaces combined with Eq. 10 gives $\|T_{\theta,\eta}(u) - T_{\theta,\eta}(v)\|_{\mathcal{X}} \le q\|u - v\|_{\mathcal{X}}$. Since B is a closed subset of the Banach space $\mathcal{X}$, it is complete. The self-mapping condition $T_{\theta,\eta}(B) \subseteq B$ and the contraction inequality invoke the Banach fixed-point theorem, yielding a unique fixed point $\bar{u} \in B$. Zero-residual consistency $G_\theta(u_\eta^\star, 0) = 0$ makes $u_\eta^\star$ a fixed point, so uniqueness gives $\bar{u} = u_\eta^\star$. Iterating the contraction inequality yields Eq. 11. □

Proof of Corollary 1. Adding and subtracting the exact correction $u_\eta^\star - u$:

$$\begin{aligned}
&\left\|T_{\theta,\eta}(u) - u_\eta^\star\right\|_{\mathcal{X}} \\
&= \left\|u - u_\eta^\star + \alpha\, G_\theta(u, \mathcal{R}_\eta(u))\right\|_{\mathcal{X}} \\
&\le (1-\alpha)\left\|u - u_\eta^\star\right\|_{\mathcal{X}} + \alpha\left\|G_\theta(u, \mathcal{R}_\eta(u)) - (u_\eta^\star - u)\right\|_{\mathcal{X}} \\
&\le [1 - \alpha(1-\rho)]\left\|u - u_\eta^\star\right\|_{\mathcal{X}}.
\end{aligned}$$

Since B is centered at $u_\eta^\star$, the update remains in B. Iterating gives the stated rate $[1 - \alpha(1-\rho)]^k$. □

Derivation of the linearization (Eq. 8). Differentiating Eq. 7 at $u_\eta^\star$:

$$DT_{\theta,\eta}(u_\eta^\star) = I + \alpha\left[D_u G_\theta(u_\eta^\star, 0) + D_r G_\theta(u_\eta^\star, 0)\, D\mathcal{R}_\eta(u_\eta^\star)\right].$$

Zero-residual consistency implies $G_\theta(u, 0) = 0$ for all u near $u_\eta^\star$, so $D_u G_\theta(u_\eta^\star, 0) = 0$. Since $\mathcal{R}_\eta(u) = -\mathcal{F}_\eta(u)$, we have $D\mathcal{R}_\eta(u_\eta^\star) = -J_\eta^\star$. Substituting yields Eq. 8. If $\|DT_{\theta,\eta}(u_\eta^\star)\| < 1$, continuity of the derivative provides a ball on which Eq. 10 holds, and Theorem 1 applies. □

Uniform convergence over a parameter family. Assume Eq. 13 holds with constant $q_0 < 1$ and $DT_{\theta,\eta}$ is uniformly continuous at the solution family: for every $\epsilon > 0$, there exists $r > 0$ such that

$$\sup_{\eta \in \mathcal{P}} \sup_{\|u - u_\eta^\star\|_{\mathcal{X}} \le r} \left\|DT_{\theta,\eta}(u) - DT_{\theta,\eta}(u_\eta^\star)\right\| < \epsilon.$$

Choosing $\epsilon < (1 - q_0)/2$ gives a uniform contraction factor $q = q_0 + \epsilon < 1$ on every ball $B_r(u_\eta^\star)$. Since each center is a fixed point, the map preserves its ball, and Theorem 1 applies simultaneously for every $\eta \in \mathcal{P}$. □

Residual–error equivalence (Eq. 14). Fix η and write $J^\star = J_\eta^\star$. By continuous Fréchet differentiability,

$$\mathcal{F}_\eta(u) = J^\star(u - u_\eta^\star) + o(\|u - u_\eta^\star\|_{\mathcal{X}}).$$

Since $J^\star$ is boundedly invertible, shrinking the neighborhood if necessary ensures the remainder satisfies $\|o(\cdot)\| \le \|u - u_\eta^\star\|/(2\|(J^\star)^{-1}\|)$. The reverse triangle inequality then gives the lower bound with $c_\eta = 1/(2\|(J^\star)^{-1}\|)$. The upper bound follows from local boundedness of $D\mathcal{F}_\eta$ and the mean-value inequality. □

C Implementation Details

C.1 Operator Implementation

The learned residual-to-correction map is used as a neural preconditioner, connecting the implementation to classical preconditioning (Benzi, 2002) and learned iterative PDE solvers (Hsieh et al., 2019; Um et al., 2020; Zhang et al., 2024). The neural preconditioner G_θ is an FNO (Li et al., 2021), following the neural-operator formulation for maps between function spaces (Kovachki et al., 2023). Its output projection predicts a correction for all T frames and is initialized to zero. For NS and RD we use width 64, 32 retained Fourier modes per spatial axis, 6 Fourier layers, and projection width 256. For the one-dimensional Burgers problem we use width 256, 64 retained modes, 16 Fourier layers, and projection width 512. The same network parameters are reused at every refinement iteration.

C.2 Training objective

Training unrolls $K = 4$ shared correction steps. The loss at iteration k combines solution reconstruction, residual reduction, contraction, and correction regularization:

$$\mathcal{L} = \sum_{k=1}^{K} \frac{2k}{K(K+1)} \left(\mathcal{L}_{\mathrm{sol}}^{(k)} + \lambda_{\mathrm{cont}} \mathcal{L}_{\mathrm{cont}}^{(k)} + \lambda_{\mathrm{corr}} \mathcal{L}_{\mathrm{corr}}^{(k)} + \lambda_{\mathrm{zero}} \mathcal{L}_{\mathrm{zero}}^{(k)} \right). \tag{19}$$

Later iterates receive linearly increasing weights. The solution term is the Charbonnier loss $\mathcal{L}_{\mathrm{sol}} = \mathrm{mean}(\sqrt{e^2 + \epsilon^2} - \epsilon)$ with $\epsilon = 10^{-3}$, The solution- and residual-contraction terms penalize an iterate when its error fails to contract by a factor of $\rho = 0.9$ relative to the preceding iterate. We use

$$(\lambda_{\mathrm{cont}}, \lambda_{\mathrm{corr}}, \lambda_{\mathrm{zero}}) = (0.1, 0.01, 0.1). \tag{20}$$

C.3 Residual Definition

We define the residual from a fourth-order spatial Crank–Nicolson discretization. Let $U^{(k)} = (u_0^{(k)}, \ldots, u_{T-1}^{(k)})$ denote the trajectory at refinement step k, and let $\mathcal{G}_{\eta,h}^{(4)}$ be the fourth-order spatial discretization of the PDE right-hand side $\partial_t u = \mathcal{G}_\eta(u)$ on the fixed grid with spacing h. In physical units, the residual is

$$R_0(U^{(k)}; \eta) = 0, \qquad R_t(U^{(k)}; \eta) = u_t^{(k)} - u_{t-1}^{(k)} - \frac{\Delta t}{2} \left[\mathcal{G}_{\eta,h}^{(4)}\left(u_t^{(k)}\right) + \mathcal{G}_{\eta,h}^{(4)}\left(u_{t-1}^{(k)}\right) \right], \quad t = 1, \ldots, T-1. \tag{21}$$

The reported residual RMS is computed from the channel-normalized residual $\widehat{R}_{t,c} = R_{t,c}/\sigma_c$, where σ_c is the training-set standard deviation of state channel c. Thus, zero residual corresponds to satisfaction of the discrete fourth-order Crank–Nicolson equation on the chosen grid; the continuous-level analysis in Section 2.3 remains an abstract operator result, and any discrepancy from the continuous PDE may include the discretization error.

C.4 Optimization and inference

Our models are trained for $20{,}000$ optimizer steps with AdamW, learning rate 10^{-4}, weight decay 10^{-4}, and gradient-norm clipping at 1. In each iteration, the first frame remains fixed. Training uses eight distributed GPUs, and a batch size of one per GPU. At inference, we initialize the state with the first frame, recompute the residual after each correction, and retain the iterate with the lowest residual RMS. Refinement runs for at most 128 iterations with damping factor $\alpha = 0.75$. It terminates when the residual has failed for eight consecutive iterations to improve by at least 10^{-3} relative to the most recent improvement reference. This criterion uses only the PDE residual and does not access the ground-truth future states.

D Detailed Dataset Setup

Common protocol. All three datasets are generated on a unit periodic domain, following periodic spectral benchmark conventions used in neural-operator studies (Li et al., 2021; Kovachki et al.,

2023). We generate $5{,}000$ trajectories for each PDE and use a fixed trajectory-level split containing $4{,}500$ training trajectories and 500 held-out test trajectories. Splitting at the trajectory level prevents temporally adjacent states from the same realization from appearing in both sets. The PDE coefficients and initial-condition family are sampled independently for each trajectory and are provided to every parameter-conditioned model. The solvers evolve the fields in double precision and store snapshots in single precision. For the $T = 32$ experiments, the first state is prescribed and the remaining 31 states are prediction targets. Per-channel mean and standard deviation are estimated using training data only and are reused for testing.

D.1 Two-dimensional Navier–Stokes flow

We consider the incompressible Navier–Stokes equation in vorticity form on $\Omega = [0, 1]^2$,

$$\partial_t \omega + \boldsymbol{u} \cdot \nabla \omega = \nu \Delta \omega + g_A(y), \qquad \nabla \cdot \boldsymbol{u} = 0, \qquad \mathrm{Re} = \nu^{-1}, \tag{22}$$

where ω is the scalar vorticity. We use Kolmogorov forcing with wave number $q = 4$; in vorticity form it is $g_A(y) = -2\pi q A \cos(2\pi q y)$. For every trajectory, we sample

$$\mathrm{Re} \sim \mathrm{LogUniform}(100, 10{,}000), \qquad A \sim \mathcal{U}(0.5, 1.0). \tag{23}$$

The initial condition is selected uniformly from seven families: a Gaussian random field, a low-frequency Fourier field, Taylor–Green modes, random vortex blobs, smoothed vorticity patches, a Brownian-bridge field, and Gaussian blobs (Herde et al., 2024). Each realization is mean-free and rescaled so that the initial root-mean-square velocity is one. The rough Brownian-bridge family is evolved for an additional burn-in time of 0.5 and then renormalized before recording.

Equation 22 is solved using a Fourier pseudo-spectral method (Trefethen, 2000) with the $2/3$ de-aliasing rule and a fourth-order integrating-factor Runge–Kutta scheme. The internal time step is selected by an advective and viscous CFL condition with CFL number 0.5 and is capped at 5×10^{-3}. We record 512 vorticity snapshots on a 256×256 grid at intervals $\Delta t = 0.01$, covering the sampled times $t \in [0, 5.11]$. Thus, a 32-frame operator window spans 0.31 physical time units.

D.2 Two-dimensional Reaction–Diffusion system

The RD benchmark is a two-species Reaction–Diffusion system on $\Omega = [0, 1]^2$,

$$\begin{aligned} \partial_t u &= D_u \Delta u - u v^2 + F(1 - u), \\ \partial_t v &= D_v \Delta v + u v^2 - (F + k) v, \end{aligned} \tag{24}$$

The reaction terms follow the classical two-species autocatalytic model (Gray & Scott, 1984). Coefficients are sampled independently as follows:

$$D_u \sim \mathrm{LogUniform}(10^{-5}, 3 \times 10^{-5}), \qquad D_v \sim \mathrm{LogUniform}(5 \times 10^{-6}, 1.5 \times 10^{-5}), \tag{25}$$

$$F \sim \mathcal{U}(0.02, 0.06), \qquad k \sim \mathcal{U}(0.045, 0.07). \tag{26}$$

We uniformly choose among single-spot, five-spot, and perturbed multi-spot initial conditions. Spot centers are sampled uniformly on the periodic domain and their Gaussian widths are sampled from $\mathcal{U}(0.02, 0.07)$. Starting from $(u, v) = (1, 0)$, each spot subtracts 0.5 times its profile from u and adds 0.25 times its profile to v. The perturbed family uses twelve spots and adds a smooth random field before clipping both concentrations to $[0, 1]$.

We solve Eq. 24 using a $2/3$-dealiased Fourier pseudo-spectral method (Trefethen, 2000) and a fixed-step fourth-order integrating-factor Runge–Kutta integrator. The internal step size is 0.25. Each trajectory contains 128 two-channel snapshots on a 128×128 grid, saved every $\Delta t = 4$ over $t \in [0, 508]$; a 32-frame window therefore spans 124 time units.

D.3 One-dimensional viscous Burgers equation

The Burgers benchmark is defined on $\Omega = [0, 1]$ as a standard viscous-flow operator-learning problem (Li et al., 2021; LeVeque, 2007):

$$\partial_t u + u\, \partial_x u = \nu\, \partial_{xx} u + f(x), \tag{27}$$

where

$$\nu \sim \mathrm{LogUniform}(10^{-3}, 10^{-1}). \tag{28}$$

With equal probability, the forcing is either absent or has the form

$$f(x) = A\sin(2\pi m x + \phi), \qquad A \sim \mathcal{U}(0.1, 1.0), \quad m \sim \mathrm{Unif}\{1,2,3,4\}, \quad \phi \sim \mathcal{U}(0, 2\pi). \tag{29}$$

The initial condition is sampled uniformly from three families: a smoothed Gaussian random field, a random superposition of the first five sine modes, and a periodic Gaussian bump with random center and width. Every initial field is made mean-free and normalized to unit root-mean-square amplitude.

Equation 27 is solved with a $2/3$-dealiased Fourier pseudo-spectral discretization (Trefethen, 2000) and fixed-step fourth-order integrating-factor Runge–Kutta integration using an internal step size of 10^{-3}. We retain 64 snapshots on a 256-point grid at intervals $\Delta t = 0.02$, corresponding to $t \in [0, 1.26]$. A 32-frame operator window spans 0.62 time units and captures the transient steepening and shock-formation regime.

E Baseline Architectures and Residual Extension

E.1 Common prediction setting

All baselines map a normalized initial field u_0 to the $T - 1 = 31$ unknown states in a $T = 32$ trajectory window. The common operator-learning setting follows standard neural-operator benchmarks (Kovachki et al., 2023). The known initial condition is inserted exactly into the resulting trajectory and is therefore not predicted. FNO (Li et al., 2021), DeepONet (Lu et al., 2021), and PINO (Li et al., 2024) produce all future states in one forward pass. IRNO (Liu et al., 2026) first produces the same one-shot FNO estimate and then updates the complete predicted trajectory for a fixed number of refinement steps. We train and evaluate a direct and a residual-extended version of every baseline under the same data split and normalization protocol.

E.2 Fourier Neural Operator and PINO

FNO (Li et al., 2021) lifts the input field and spatial coordinates to a latent representation, applies alternating Fourier convolution and pointwise linear layers, and projects the result to all 31 target states. For NS and RD, we use 6 Fourier layers with width 64, retain 32 Fourier modes per spatial axis, and use a projection width of 128. Burgers uses width 256, 64 retained modes, 16 Fourier layers, and projection width 512.

PINO (Li et al., 2024) uses the identical FNO architecture but minimizes a combination of data and physics losses,

$$\mathcal{L}_{\mathrm{PINO}} = \lambda_{\mathrm{data}}\mathcal{L}_{\mathrm{Charb}} + \lambda_{\mathrm{phys}}\mathcal{L}_{\mathrm{PDE}}, \qquad \lambda_{\mathrm{data}} = \lambda_{\mathrm{phys}} = 1. \tag{30}$$

Here $\mathcal{L}_{\mathrm{Charb}}$ is the supervised Charbonnier loss and $\mathcal{L}_{\mathrm{PDE}}$ is the mean squared residual of the predicted trajectory, excluding the prescribed initial state. We use a Crank–Nicolson residual for NS and forward-difference equation residuals for RD and Burgers. The residual variant retains this objective and additionally uses the residual features [2].

E.3 DeepONet

Our DeepONet (Lu et al., 2021) implementation uses a convolutional branch network and a temporal trunk network. The branch maps the input field to 64 spatial basis coefficients per output channel, while a 2-hidden-layer MLP maps each normalized query time to the corresponding temporal basis. Their inner product gives the predicted field at each of the 31 target times. The branch width is 32 and both trunk hidden layers have width 128. In the residual variant, the branch receives additional residual input while the temporal trunk remains unchanged.

[2] There are two types of residuals here. The first is the equation residual mentioned here; it is calculated from the PINO output and serves as the loss function, consistent with (Li et al., 2024). The second is the Crank-Nicolson residual described in Appendix C.3; acting as a residual extension for the baseline methods, and is calculated from the input.

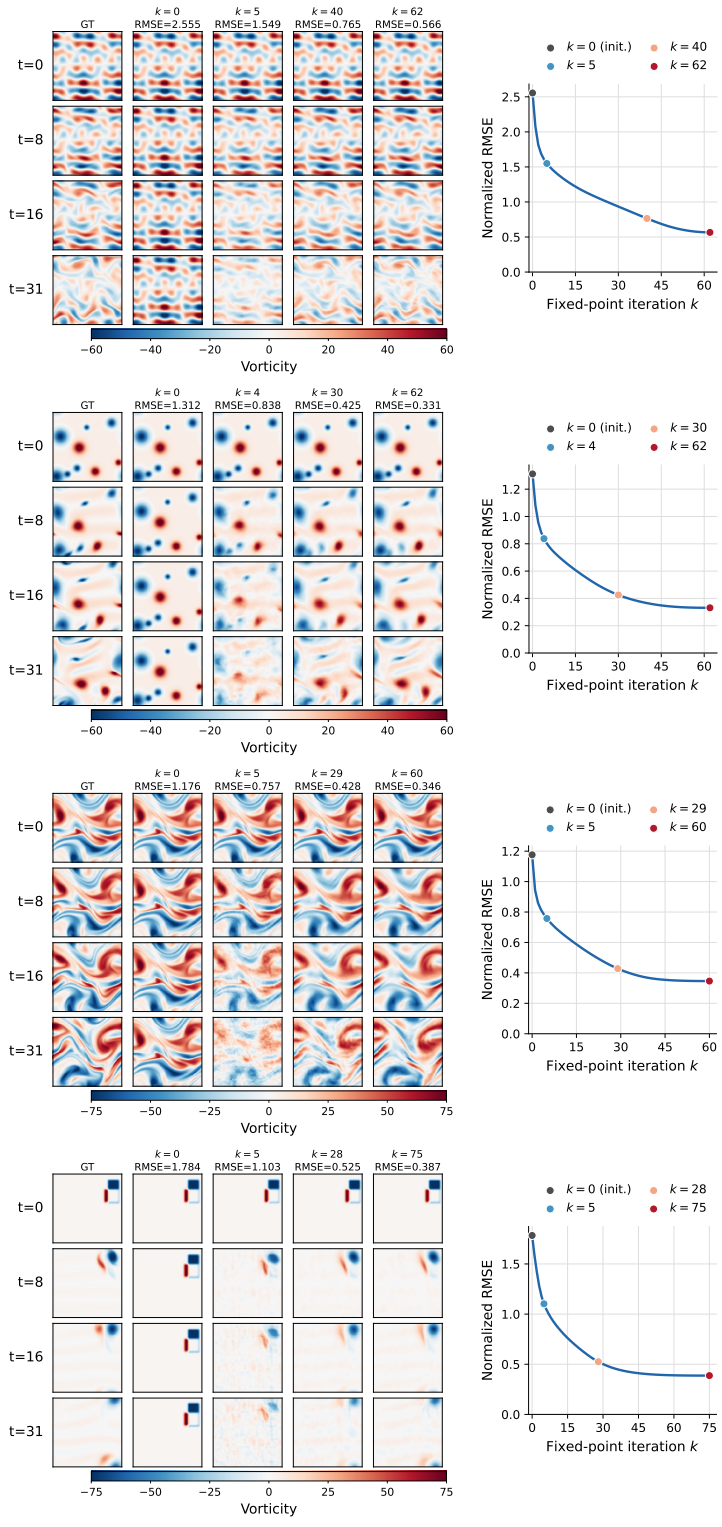

Figure 6: **Additional Navier–Stokes fixed-point trajectories.** Four held-out examples spanning different Reynolds numbers and forcing amplitudes are shown using the same layout as Figure 2. Across cases, later iterates recover progressively finer vorticity structure and reduce normalized RMSE; the selected iteration varies by instance.

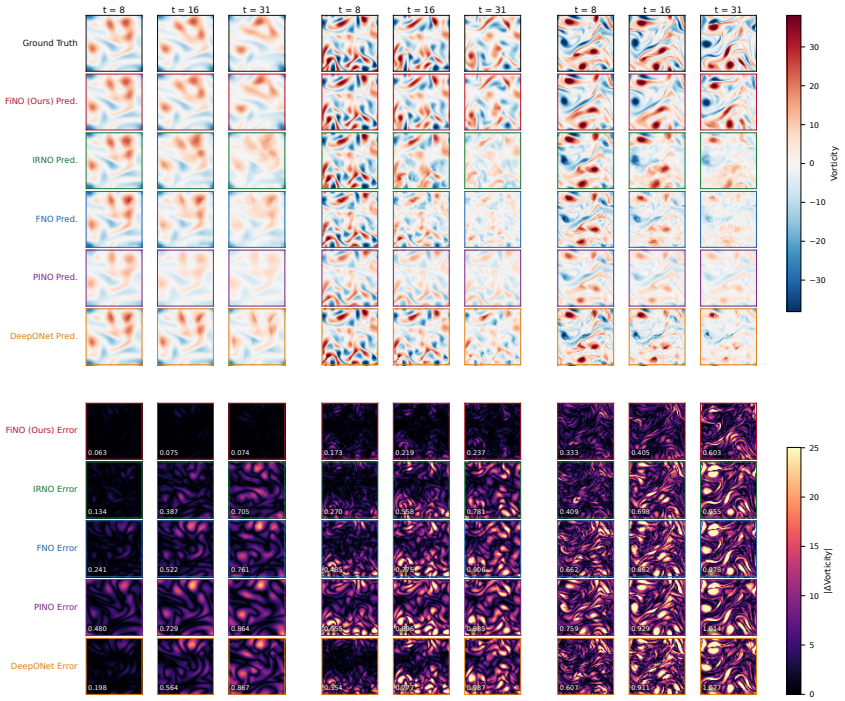

Figure 7: **Navier–Stokes predictions across the $T = 32$ window.** Ground truth, predictions, and absolute vorticity errors are shown at $t \in \{8, 16, 31\}$ for three representative held-out cases. Numbers in the error panels report physical relative L_2 error. All baseline visualizations use the residual-extended variants.

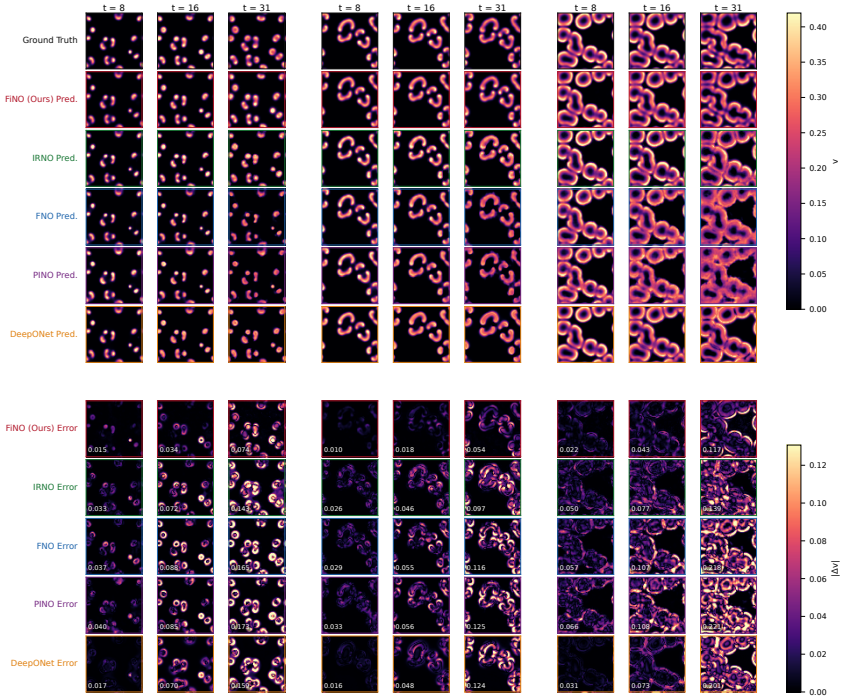

Figure 8: **Reaction–Diffusion predictions across the $T = 32$ window.** Ground truth, predictions, and absolute errors in the v channel are shown at $t \in \{8, 16, 31\}$ for three representative held-out cases. Numbers in the error panels report physical relative L_2 error. All baseline visualizations use the residual-extended variants.

Table 3: Conditioning ablation with $T = 32$. Lower is better.

Dataset	Conditioning	RMSE	Res. RMS
Navier–Stokes	State only	0.5522	0.0683
	Parameters	0.4928	0.0728
	Residual	**0.1808**	**0.0239**
Burgers	State only	0.5305	0.1618
	Parameters	0.3078	0.1399
	Residual	**0.2034**	**0.0881**
Reaction–Diffusion	State only	0.2991	0.0501
	Parameters	0.2692	0.0736
	Residual	**0.2098**	**0.0197**

Table 4: Inference efficiency on the Navier–Stokes test set.

Metric	Value
Mean iteration time	8.62 ms
Residual evaluation time	1.82 ms
Operator forward time	6.82 ms
Residual share	21.3%
Mean num. iterations	38.66 ± 9.82
Mean solving time	333.24 ± 84.64 ms
Peak memory	592.8 MiB

E.4 IRNO

IRNO (Liu et al., 2026) wraps a pretrained FNO and refines its complete trajectory estimate using a shared correction network. Starting from $h^{(0)} = \mathrm{FNO}(x)$, the update is

$$h^{(k+1)} = h^{(k)} + \alpha \Psi_\theta(x, h^{(k)}), \qquad \alpha = 0.2. \tag{31}$$

The FNO is frozen during IRNO training. The correction Ψ_θ is a four-layer periodic convolutional network of width 32, shared across 6 refinement steps. Training supervises every refined estimate, assigns larger weights to later iterations, progressively emphasizes higher spatial frequencies, and penalizes a nonzero correction at the ground-truth fixed point. Direct IRNO wraps a Direct FNO, whereas residual-extended IRNO wraps a residual-extended FNO and includes the residual features in the refinement context.

E.5 Optimization

All baselines are trained for 20,000 optimizer steps using AdamW with learning rate 10^{-3}, weight decay 10^{-4}, and gradient norm clipping at 1. Training uses eight GPUs. The supervised models use a Charbonnier loss with smoothing parameter 10^{-3}; the PINO and IRNO objectives add the terms described above. Evaluation always uses the same non-overlapping $T = 32$ windows.

F Additional Results

F.1 Qualitative Evaluation

Figure 6 extends the two representative Navier–Stokes trajectories shown in the main text with four additional held-out cases spanning Reynolds numbers and forcing amplitudes.

Figures 7 and 8 provide a spatial view of the aggregate results in Table 1. Ground-truth states, predictions, and absolute-error maps are shown at $t \in \{8, 16, 31\}$ for representative held-out $T = 32$ windows; numbers in the error maps report the physical relative L_2 error of the displayed frame. The direct and iterative baselines tend to blur or displace fine vortical structures in Navier–Stokes and

Table 5: Navier–Stokes ($T = 32$) performance for different residual refresh intervals. Lower is better.

Residual interval	RMSE ↓	Res. RMS ↓	Mean iter.	Max iter.	Early stop
1	0.1851	0.0244	38.7	86	100.0%
2	0.2283	0.0245	58.2	128	99.8%
4	0.2825	0.0318	75.6	128	70.4%

accumulate error around reaction fronts in the Reaction–Diffusion system. FiNO preserves more of the reference structure, with remaining errors concentrated near the most rapidly varying structures. This behavior is consistent with residual-driven corrections targeting local equation residual instead of replacing the entire trajectory in one pass.

F.2 Conditioning Ablation

The conditioning ablation in Table 3 compares state-only, explicit parameter and residual conditioning while keeping the correction backbone and optimization protocol fixed. Parameter definition follows the dataset setup in Appendix D. Residual conditioning achieves lower solution error and residual RMS on all three PDEs. These results suggest that the residual provides a more useful instance-specific signal than either the current state alone or the parameter vector alone.

F.3 Inference Efficiency

Table 4 reports the inference-time breakdown, iteration statistics, and peak memory usage of FiNO on the Navier–Stokes test set. The residual evaluation time is 21.3% of the total iteration time, and the peak memory usage is 592.8 MiB. The evaluation is performed using the $T = 32$ FiNO checkpoint on an NVIDIA A100 40GB GPU.

F.4 Effect of Residual Refresh Frequency

To determine whether FiNO benefits from residual-driven solver dynamics rather than simply applying the correction network more times, we vary the residual refresh interval while applying the neural correction at every iteration. An interval of s recomputes the residual every s correction steps and reuses the previous residual otherwise. All settings use the same residual-based stopping rule and maximum iteration budget of 128. Each model is trained with its corresponding residual refresh schedule.

Table 5 shows that less frequent residual refresh leads to more correction iterations, yet poorer solution accuracy and residual quality. Thus, repeatedly applying the correction network with stale residual information cannot substitute for fresh equation feedback; FiNO's advantage comes from coupling iterative correction with updated residual information for faster and more reliable convergence.