Solver-as-Teacher: Solver-Guided On-Policy Post-Training Framework for PDE Foundation Models

Hao Wei
Technical University of Munich
hao.wei@tum.de

Shangzhe Li
UNC Chapel Hill
shangzhe@unc.edu

Nils Thuerey
Technical University of Munich
nils.thuerey@tum.de

Abstract

PDE foundation models (PDE-FMs) promise general-purpose surrogate simulation, yet deploying a pretrained model to a new physical regime inevitably introduces distribution mismatch, demanding task-specific post-training. The dominant approach, supervised fine-tuning (SFT), is fundamentally off-policy: the model trains on pre-generated states but is evaluated autoregressively on its own. In chaotic regimes, this mismatch is catastrophic as per-step errors compound exponentially along the Lyapunov spectrum, and exact pointwise trajectory matching beyond the predictability horizon becomes an ill-posed training signal. To resolve this, we introduce **Solver-as-Teacher (SaT)**, an on-policy post-training framework for PDE-FMs. During training, the student rolls out autoregressively; a numerical solver branches multi-step correction targets from each student-visited state; and the student is updated against these solver corrections, learning to correct its own rollout dynamics. We instantiate SaT in a unified post-training suite with SFT, physics-informed temporal alignment, and learned-teacher on-policy distillation. Across seven 2D/3D PDE benchmarks and six pretrained PDE-FMs, SaT variants achieve the lowest 50-step rollout RMSE in 23 of 24 in-distribution model–task pairs. The best SaT variant reduces RMSE over SFT up to 90.0%, and SaT gives the lowest RMSE on all seven OOD parameter-shift benchmarks.

1 Introduction

The solution of partial differential equations (PDEs) is a cornerstone of modern scientific computing. A rapidly growing body of work trains *PDE foundation models* (PDE-FMs): large neural surrogates pretrained on broad simulation data that learn a time-stepping operator $\mathcal{G}_\theta : u(t) \mapsto u(t+\Delta t)$ and can be composed autoregressively at inference [1–9]. Once pretrained, PDE-FMs promise to generalize across diverse physical systems, positioning them as a general engine for scientific computing.

This promise, however, rests on the assumption that pretraining covers the deployment distribution. Systems governed by PDEs differ across physical regimes: boundary conditions, initial conditions, or physical parameters such as the Reynolds number. But even slight mismatches push the model into state-space regions it has rarely seen during pretraining [10, 11], leading to distribution mismatch. Moreover, there is a fundamental challenge for chaotic regimes: small per-step errors are amplified exponentially along the Lyapunov spectrum, so trajectories diverge from the physical solution manifold, degrade into spectral blur, corrupt attractor statistics, or blow up entirely, even for models with excellent short-horizon metrics [12–17]. Therefore, deploying a pretrained PDE-FM to a new scenario or simply extending the rollout horizon demands task-specific adaptation with proper post-training; without it, long-horizon fidelity is not merely degraded but infeasible.

Yet while pretraining for PDE-FMs has matured rapidly, their post-training methods have not. The most common approach remains supervised fine-tuning (SFT), where the model is trained to minimize pointwise discrepancies between its predictions and reference trajectories. This objective is natural

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

for short-horizon accuracy but fundamentally *off-policy*: the model is optimized on states sampled from a fixed, pre-generated dataset, whereas at inference, it is evaluated autoregressively on states generated by its own previous predictions. This creates a distribution mismatch [18] between the trajectories seen during training and those generated by the PDE-FMs during inference. The issue is more severe for chaotic and turbulent systems, where small local errors are exponentially amplified by positive Lyapunov exponents, leading to long-horizon distribution drift [19–21]. In such regimes, exact pointwise trajectory matching beyond the predictability horizon becomes an ill-posed training signal: two physically valid trajectories may rapidly decorrelate pointwise while still sharing the same governing dynamics [13, 20]. Thus, post-training PDE-FMs should not only fit reference trajectories, but learn to recover **physically meaningful dynamics** from the states produced by their own rollouts.

Although post-training algorithms beyond SFT for LLMs have recently flourished [22], most do not directly apply to PDE-FMs. Current PDE-FMs are predominantly deterministic neural operators: given a continuous field state, they output a single next-state prediction rather than a probability distribution over tokens, making standard generative alignment techniques inapplicable. Nor does on-policy distillation [23] with a larger FM as teacher solve the problem. It closes the distribution gap but inherits the teacher's own approximation errors: in chaotic regimes where per-step errors grow exponentially, the teacher FM's hallucinations transfer directly to the student [24, 25].

To address this gap, we introduce **Solver-as-Teacher (SaT)**, a solver-guided on-policy post-training framework designed explicitly for PDE-FMs. We keep the core idea of on-policy distillation: train the model on the continuous states it actually visits during autoregressive rollout, and provide dense expert supervision at those states. SaT uses a numerical solver $\mathcal{S}$ as the teacher. Unlike a learned teacher, the solver provides high-fidelity physical correction targets governed by the underlying PDE, up to discretization and time-stepping error. This enables dense supervision not only on the student's own state distribution, but also ensures the supervision is aligned with the correct physical dynamics. As shown in fig. 1, SaT lets the student drive training: (1) at every iteration, the student rolls out autoregressively $\hat{u}_1, \ldots, \hat{u}_K$ from initial state u_0; (2) the numerical solver branches forward from each student-visited state $\hat{u}_k$ to produce multi-step correction targets $\mathcal{S}^j(\hat{u}_k)$; and (3) the student parameters are updated to minimize the discrepancy between its predictions and the solver-induced dynamics at the states the student actually visits. In this way, SaT converts post-training from static trajectory fitting into solver-guided correction of the model's own rollout dynamics.

Building on this core mechanism, we further cast SaT as one element of a *unified post-training suite* that instantiates complementary training objectives within the same infrastructure: **On-Policy Distillation (OPD)** reuses the on-policy loop but replaces the solver with a frozen, larger foundation model as teacher [23], isolating the contribution of the solver-teacher signal; and **PITA** [26] applies physics-informed PDE residuals as a self-supervised signal for trajectory alignment. Together with standard SFT as the off-policy baseline, SaT, OPD, and PITA form a coherent spectrum of post-training strategies, enabling the first systematic comparison for PDE foundation models. Our main contributions are summarized as follows:

- **Solver-as-Teacher on-policy distillation.** We propose SaT, the first on-policy distillation framework for PDE foundation models that uses a numerical solver as the teacher, training the student on its own autoregressive trajectory with dense, per-step solver-derived corrections.
- **Unified post-training suite for PDE-FMs.** We integrate SFT (standard baseline), OPD (larger-FM on-policy distillation), SaT, and PITA (physics-informed trajectory alignment) into a single framework, enabling the first systematic comparison of post-training strategies for PDE-FMs.
- **Extensive evaluation on diverse PDEs and backbones.** We evaluate the suite on seven 2D/3D PDE benchmarks using six pretrained PDE-FM backbones, demonstrating that SaT variants achieve the best 50-step rollout RMSE in 23 of 24 in-distribution model–task pairs and reduce aggregate RMSE over SFT by 29.1%–90.0%. Additional experiments extend SaT to 96^3 resolution and to a second solver family, a finite-volume solver for wall-bounded turbulent channel flows.

2 Related Work

PDE Foundation Models. The success of the first PDE-FM [27] has spurred a wave of foundation models. MPP [3] established the multi-physics pretraining paradigm, and its successor Walrus [5] scales to 1.3B parameters. Poseidon [2] introduces a multiscale operator transformer (scOT) with time-conditioned LayerNorm, enabling a single model to handle variable time horizons. DPOT [4]

adopts a Fourier-attention transformer pre-trained with noise-perturbed inputs on a large collection of PDE simulations. PDE-Transformer [1] proposes a Swin-style architecture pretrained on 16 PDE types, while its 3D extension P3D [8] handles up to 512^3 resolution via windowed attention with global context fusion. OmniFluids [6] combines physics-only pretraining, operator distillation, and few-shot fine-tuning through a mixture-of-operators routing mechanism. MORPH [7] supports arbitrary data modalities from 1D to 3D via axial attention with native LoRA support. Recent studies focus on generalization and robustness during inference. Some incorporate PDE residuals into pretraining [28], while others leverage In-Context Learning for PDE-FMs [29–31]. Others build physical symmetries, such as rotational equivariance, directly into transformer backbones for neural PDE solvers [32]. Despite this architectural diversity, most PDE-FMs remain vulnerable to distribution drift between training and inference, especially in chaotic regimes where per-step errors compound exponentially, causing trajectories to diverge from the physical attractor [12, 13, 16, 17].

Remedies for Distribution Drift. Remedies for distribution drift have been proposed prior to the current wave of PDE-FMs. The pushforward trick [33] trains on multi-step unrolled predictions to reduce exposure bias; Solver-in-the-Loop [34] leverages a differentiable solver to advance the solution in time; INC [21] improves stability of hybrid methods by coupling corrections into the governing equations, and PhysicsCorrect [35] applies solver-residual corrections at inference without updating weights. For statistical fidelity in chaotic regimes, DySLIM [36] and Jiang et al. [20] propose invariant measure matching losses, while Li et al. [13] demonstrate that Sobolev loss with dissipativity regularization enables stable long-horizon emulation. Other remedies, such as multi-model training with multi-timescales [37], noise injection [38], and multiresolution targets [39], reduce errors with increased complexity and, more importantly, are difficult to transfer to PDE-FM post-training.

Post-training methods. Post-training has become a central tool for improving foundation models, including supervised fine-tuning, RL-based alignment, and knowledge distillation [22, 40–42]. However, these methods are not directly transferable to PDE-FMs. Most PDE-FMs are deterministic, mapping a continuous field to a single next-state prediction rather than producing token-level probability distributions. Thus, LLM-style objectives based on log probabilities, policy ratios, or stochastic sampling require substantial reformulation. Moreover, as long-horizon PDE prediction is sensitive to error accumulation, distillation from a larger teacher may propagate the teacher's own approximation errors [23–25]. This gap has motivated post-training methods tailored to PDE-FMs. LoRA-based parameter-efficient adaptation [43] has been explored [1, 7], but such methods are typically offline fine-tuning procedures. F-Adapter allocates adapter capacity according to spectral complexity [44], and Spectral-Refiner fine-tunes FNO spectral layers using a Sobolev norm [45]; however, these designs are closely tied to spectral architectures and are less directly applicable to transformer-based PDE-FMs. Mansingh et al. [46] introduce inference-time scaling, but rely on converting deterministic PDE-FMs into stochastic samplers via dropout at inference. PITA incorporates physics-informed constraints into self-supervised trajectory alignment [26], yet physics residuals serve as soft constraints and, by themselves, do not guarantee long-horizon rollout performance.

3 Method: Solver-as-Teacher

We now formalize the SaT framework. We first establish notation, then describe the multi-step solver branching mechanism, efficient dispatch strategies, and the training objective.

Preliminaries Consider a time-dependent PDE on a periodic spatial domain Ω:

$$\frac{\partial u}{\partial t} = \mathcal{F}(u, \nabla u, \nabla^2 u, \ldots; \mu), \qquad u(x,0) = u_0(x), \tag{1}$$

where $\mathcal{F}$ is the spatial operator and μ denotes PDE parameters. A *student* FM $\mathcal{G}_\theta \colon u(t) \mapsto u(t+\Delta t)$ is applied autoregressively to produce a K-step rollout $\hat{\tau} = (\hat{u}_0, \hat{u}_1, \ldots, \hat{u}_K)$ with $\hat{u}_{i+1} = \mathcal{G}_\theta(\hat{u}_i)$. A *teacher* numerical solver $\mathcal{S}$ maps any state $\hat{u}_i$ to the next physical state $\mathcal{S}(\hat{u}_i)$, accurate up to time-stepping and spatial discretization error. Given this setup, the central question is how the teacher solver provides correction targets to the student during on-policy rollout. We write $\mathcal{S}^j(\hat{u}_k)$ for j sequential solver applications starting from $\hat{u}_k$. Throughout, we use K for the rollout horizon and $m_{\max}$ for the lookahead depth of the teacher.

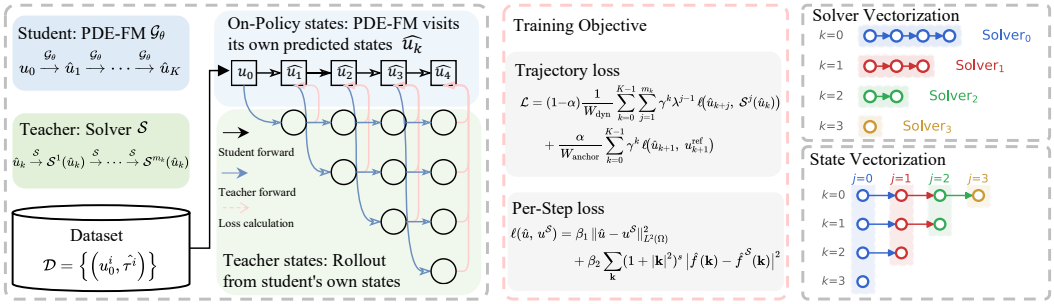


Figure 1: **SaT overview.** *Left:* the student FM $\mathcal{G}_\theta$ autoregressively rolls out on-policy states $\hat{u}_1, \ldots, \hat{u}_K$ from an initial condition u_0; the teacher $\mathcal{S}$ branches multi-step correction targets from each student-visited state, as $\mathcal{S}^j(\hat{u}_k)$ for $j = 1, \ldots, m_k$. *Center:* the combined training objective from eq. (3) consists of (i) a dynamics, on-policy multi-step loss that compares the FM's autoregressive predictions against solver-generated chains, and (ii) an anchor loss that directly penalizes deviation from a pre-computed reference trajectory u^{ref}_{k+1}. The per-step loss in eq. (4) is a weighted combination of an L^2 pointwise error and a Sobolev term. *Right:* two vectorization strategies for the multi-step solver branching. **Solver vectorization**: each branch k is assigned to an independent solver instance, computing its full chain in parallel. **State vectorization**: all active branches are concatenated into a mega-batch at each depth level j and processed in a single vectorized solver call.

Multi-Step Solver Branching From each student-visited state $\hat{u}_k$, the teacher branches forward m_k steps, producing a tree of correction targets (fig. 1, left):

$$m_k = \min(m_{\max},\ K-k), \qquad \mathcal{S}^j(\hat{u}_k) \ \text{ for } \ j = 1, \ldots, m_k. \tag{2}$$

The natural truncation gives longer lookahead early in the trajectory, where the student is closer to the physical manifold and solver targets are most reliable, and shorter lookahead toward the end, where the student has drifted further. Setting $m_{\max} = 1$ recovers one-step on-policy distillation; with $m_{\max} > 1$, the student receives supervision not just on its immediate output but on how its predictions compound over multiple steps. Crucially, the teacher is treated as a black-box oracle: no gradient flows through $\mathcal{S}$. This permits the use of any solver without differentiability requirements, unlike hybrid pipelines [34] that backpropagate through the full computation graph of the solver. The black-box property also enables flexible dispatch strategies for the solver evaluations.

Efficient Solver Dispatch The multi-step branching structure requires $\sum_{k=0}^{K-1} m_k$ solver evaluations per training step, each applied to a mini-batch of B samples. We implement two complementary vectorization strategies: **(1) Solver vectorization** assigns each branch k to an independent solver instance that computes its full chain $[\mathcal{S}^1(\hat{u}_k), \ldots, \mathcal{S}^{m_k}(\hat{u}_k)]$ in a single fused dispatch. Because each solver instance processes one branch independently, this strategy does not require the solver to support batched evaluation—any black-box time-stepper suffices. **(2) State vectorization** iterates over depth level j; at each level, all branches that have not yet terminated are concatenated into a single mega-batch and processed via one batched solver call. This strategy exploits solvers that natively support vectorized batch evaluation (e.g. via `jax.vmap`), amortizing dispatch overhead across all active branches at each depth level and maximizing GPU occupancy. The two variants are shown in fig. 1, right. State vectorization consistently achieves the highest throughput in our benchmarks ($\sim$13$\times$ over sequential execution; see appendix C), while the PyTorch–JAX bridge cost is in appendix B. We use state vectorization as the default backend throughout all experiments.

Training Objective With the solver targets efficiently computed, we now define how the student is trained against them. Analogous to OPD in language models [23], we formulate a training objective that combines on-policy solver correction with an optional off-policy reference anchor. At each step k of the student's rollout, two supervision signals are available: (1) the *solver correction* $\mathcal{S}(\hat{u}_k)$, computed on-policy from the student's own state, providing the physically correct next state from wherever the student currently is; and (2) the *reference trajectory* u^{ref}_{k+1}, the solver-derived reference state from a pre-computed trajectory, providing an off-policy anchor toward the reference solution path.

The full training objective combines a dynamics loss $\mathcal{L}_{\text{dyn}}$ from multi-step solver corrections with an anchor loss $\mathcal{L}_{\text{anchor}}$ from reference trajectories:

$$\mathcal{L} = (1-\alpha)\frac{1}{W_{\text{dyn}}}\sum_{k=0}^{K-1}\sum_{j=1}^{m_k}\gamma^k\lambda^{j-1}\,\ell\big(\hat{u}_{k+j},\,\mathcal{S}^j(\hat{u}_k)\big) + \frac{\alpha}{W_{\text{anchor}}}\sum_{k=0}^{K-1}\gamma^k\ell\left(\hat{u}_{k+1},\,u^{\text{ref}}_{k+1}\right), \quad (3)$$

where $\gamma, \lambda \in (0, 1]$ are decay weights for trajectory position and solver branch depth, $\alpha \in [0, 1]$ interpolates between pure on-policy correction and reference-trajectory supervision, and $W_{\text{dyn}}, W_{\text{anchor}}$ are normalization constants that ensure a stable learning rate across rollout configurations. Different values of α yield the key training methods as special cases:

- $\alpha = 0$: **SaT-0**, pure on-policy solver correction. The student learns exclusively from the teacher's evaluation of its own states.
- $\alpha = 0.5$: **SaT-.5**, interpolating between on-policy solver correction and off-policy reference supervision. The dynamics loss teaches the model to recover from its own errors; the anchor loss guides it back toward the reference trajectory.
- $\alpha = 1$: **SaT-1**, which reduces to uniformly weighted unrolled supervised fine-tuning [34], with no on-policy correction signal.

We recommend $\alpha = 0$ (SaT-0) as the default, especially for chaotic systems: it gives the lowest cross-model RMSE on six of seven tasks, is within 1.2% of SaT-.5 on the seventh (KS3D), and is the best method in 10 of the 14 model–task pairs on the chaotic KS and KF tasks (tables 1 and 22).

Per-step composite loss. The per-step loss $\ell(\cdot, \cdot)$ combines pointwise and spectral supervision:

$$\ell(\hat{u},\, u^{\mathcal{S}}) = \beta_1 \cdot \|\hat{u} - u^{\mathcal{S}}\|^2_{L^2(\Omega)} + \beta_2 \cdot \sum_{\mathbf{k}}(1 + |\mathbf{k}|^2)^s \left|\hat{f}(\mathbf{k}) - \hat{f}^{\mathcal{S}}(\mathbf{k})\right|^2, \quad (4)$$

where β_1, β_2 weight the pointwise and spectral terms, and $\hat{f}$, $\hat{f}^{\mathcal{S}}$ denote the Fourier coefficients of the student and solver outputs, respectively. The Sobolev term penalizes high-frequency discrepancies via the H^s norm [45], computed exactly in Fourier space using Parseval's identity. Together, the unified objective and composite per-step loss define the complete SaT training procedure.

4 Experimental Configurations

Baselines We integrate SaT with different values of α as described in section 3 into a unified post-training suite alongside three complementary methods, enabling the first systematic comparison of post-training strategies for PDE-FMs. **SFT** (supervised fine-tuning) serves as the baseline; it is widely applied in existing fine-tuning of PDE-FMs [2–4]. **On-Policy Distillation (OPD)** [23] reuses the same on-policy loop as SaT-0, but replaces the numerical solver with a fine-tuned, larger foundation model as teacher, isolating the contribution of solver corrections versus approximate FM corrections. **PITA** [26] (Physics-Informed Temporal Alignment) applies physics-informed constraints as a self-supervised signal for trajectory alignment: the student's autoregressive rollout is penalized by PDE residuals computed from its own predictions, encouraging temporal physical consistency without requiring solver labels at training time. All methods share the same trainer, configuration system, and evaluation engine; they differ only in the teacher source and loss formulation.

PDE Benchmarks We evaluate on seven PDE tasks spanning both 2D and 3D, including chaotic and turbulent regimes. All 2D cases are sourced from APEBench [47], and the 3D cases are generated at higher resolution (64^3) by directly instantiating the same Exponax solver used by APEBench. The three 2D tasks are discretized at 160^2 resolution on $\Omega = [0, 2\pi]^2$: **Kuramoto–Sivashinsky (KS2D)**, a scalar fourth-order PDE exhibiting spatiotemporal chaos through the competition of anti-diffusion and hyper-diffusion; **Burgers (Burgers2D)**, a scalar conservation law with viscous diffusion that develops sharp fronts and shock structures; and **Kolmogorov Flow (KF2D)**, forced incompressible Navier–Stokes in vorticity formulation, driven by a sinusoidal body force at mode 4. The four 3D tasks are discretized at 64^3 resolution on $\Omega = [0, 2\pi]^3$: **KS3D** and **Burgers3D** extend their 2D counterparts to three spatial dimensions; **Kolmogorov Flow (KF3D)**, forced 3D incompressible Navier–Stokes in velocity formulation (3 channels, 200-step warm-up to reach statistical stationarity); and **Decaying Turbulence (DT3D)**, unforced 3D incompressible Navier–Stokes in velocity formulation (3 channels),

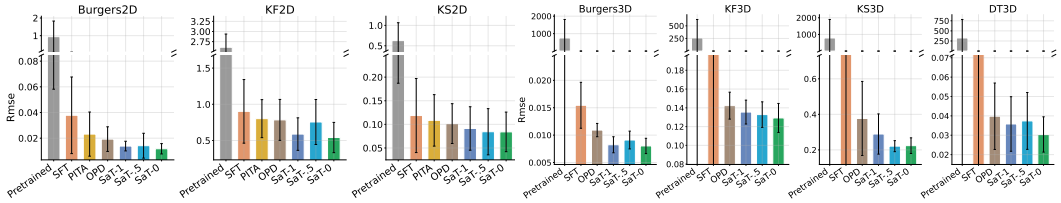
where energy cascades freely from large to small scales without external forcing. We additionally evaluate on out-of-distribution (OOD) zero-shot test sets to probe generalization; dataset specifications are provided in appendix E.4. The governing equations, complete solver parameter tables, and data splits are provided in appendix E. Results at 96^3 resolution and on wall-bounded turbulent channel flow (TCF3D) with a second solver are reported in appendices H and L.

Models and Teacher We evaluate SaT across six pretrained PDE foundation models that span diverse architecture families and cover both 2D and 3D spatial domains. **PDE-S** [1] (33.2M params) is a Swin-like U-Net Transformer with hierarchical shifted-window attention. **MPP-B** [3] (115.6M params) employs an Axial Vision Transformer (AViT) with interleaved spatial and temporal axial attention, pretrained across heterogeneous physical systems. **DPOT-Ti** [4] uses Adaptive Fourier Neural Operator (AFNO) spectral mixing blocks that apply learnable complex-valued MLPs to retained Fourier modes; it is the only model applied to both 2D (5.2M params) and 3D (12.7M params), where the 3D instance keeps the same DPOT-Ti block width/depth but replaces the 2D patch embedding, positional embedding, and output reconstruction with 3D counterparts. **Poseidon-T** [2] (20.7M params) builds on ScOT, a SwinV2-style scalable operator transformer with built-in spectral resolution adaptation. For 3D, **P3D-S** [8] (11.2M params) combines a Conv3D U-Net encoder/decoder with windowed 3D Transformer blocks, and **MORPH-Ti** [7] (9.9M params) uses 4D axial attention decomposed along the temporal, depth, height, and width axes with inter-field cross-attention. All student models are fine-tuned with full weights in our experiments. Detailed model configurations are provided in appendix D. The teacher solver is Exponax [47], a JAX-based ETDRK pseudo-spectral solver. We deliberately choose a JAX solver within a PyTorch training pipeline to demonstrate that SaT is agnostic to the solver backend. To test a second solver family, appendix L replaces Exponax with PICT [48], a PyTorch finite-volume PISO solver, on wall-bounded turbulent channel flow. For OPD, the teacher is PDE-L (518M params, 2D) or P3D-L (181M params, 3D), fine-tuned before being used for training the students. Experimental details and training hyperparameters are provided in appendix E; end-to-end wall-clock benchmarks comparing SaT, SFT, and OPD are in appendix A; the JAX↔PyTorch bridge overhead is characterized in appendix B; and the efficiency between state vectorization and solver vectorization is detailed in appendix C.

5 Results

Table 1: Cross-model aggregate diagnostics on in-distribution test sets for 50 steps, across 7 PDE benchmarks. Each entry is the mean ± std over evaluated architectures (4 for 2D, 3 for 3D); lower is better. Best per row **bolded**. RMSE, Sobolev H^1, and normalized energy-spectrum errors are defined in appendix F; per-model RMSE values are reported in table 22. PITA does not support 3D tasks (–).

Task	Metric	SFT	PITA	OPD	SaT-1	SaT-.5	SaT-0	N models
Burgers2D	RMSE	0.0378 ± 0.0298	0.0232 ± 0.0172	0.0193 ± 0.0096	0.0138 ± 0.0038	0.0141 ± 0.0097	$\mathbf{0.0118 \pm 0.0040}$	4
	H^1	0.0380 ± 0.0300	0.0232 ± 0.0172	0.0193 ± 0.0096	0.0139 ± 0.0038	0.0142 ± 0.0097	$\mathbf{0.0118 \pm 0.0041}$	
	Spectrum	0.0502 ± 0.0576	0.0234 ± 0.0263	0.0223 ± 0.0163	0.0090 ± 0.0025	0.0150 ± 0.0194	$\mathbf{0.0077 \pm 0.0059}$	
KF2D	RMSE	0.9023 ± 0.4394	0.8017 ± 0.2627	0.7845 ± 0.2827	0.5884 ± 0.2246	0.7554 ± 0.3107	$\mathbf{0.5409 \pm 0.2089}$	4
	H^1	0.9032 ± 0.4399	0.8024 ± 0.2629	0.7852 ± 0.2829	0.5890 ± 0.2248	0.7561 ± 0.3110	$\mathbf{0.5414 \pm 0.2091}$	
	Spectrum	0.0250 ± 0.0135	0.0169 ± 0.0090	0.0179 ± 0.0009	0.0166 ± 0.0060	0.0210 ± 0.0098	$\mathbf{0.0126 \pm 0.0054}$	
KS2D	RMSE	0.1187 ± 0.0784	0.1084 ± 0.0545	0.1016 ± 0.0422	0.0913 ± 0.0460	0.0844 ± 0.0489	$\mathbf{0.0841 \pm 0.0419}$	4
	H^1	0.1191 ± 0.0787	0.1088 ± 0.0547	0.1019 ± 0.0423	0.0916 ± 0.0462	0.0847 ± 0.0491	$\mathbf{0.0843 \pm 0.0421}$	
	Spectrum	0.0778 ± 0.0733	0.0438 ± 0.0203	0.0440 ± 0.0171	0.0338 ± 0.0121	0.0365 ± 0.0050	$\mathbf{0.0307 \pm 0.0131}$	
Burgers3D	RMSE	0.0154 ± 0.0042	–	0.0109 ± 0.0012	0.0082 ± 0.0015	0.0091 ± 0.0016	$\mathbf{0.0080 \pm 0.0014}$	3
	H^1	0.0157 ± 0.0043	–	0.0110 ± 0.0012	0.0083 ± 0.0014	0.0092 ± 0.0016	$\mathbf{0.0081 \pm 0.0014}$	
	Spectrum	0.0562 ± 0.0370	–	0.0290 ± 0.0100	0.0227 ± 0.0003	0.0201 ± 0.0106	$\mathbf{0.0180 \pm 0.0072}$	
KF3D	RMSE	0.6987 ± 0.7455	–	0.1423 ± 0.0143	0.1366 ± 0.0115	0.1328 ± 0.0136	$\mathbf{0.1292 \pm 0.0155}$	3
	H^1	0.7142 ± 0.7662	–	0.1431 ± 0.0143	0.1374 ± 0.0115	0.1336 ± 0.0137	$\mathbf{0.1300 \pm 0.0155}$	
	Spectrum	0.1431 ± 0.1788	–	0.0616 ± 0.0693	$\mathbf{0.0250 \pm 0.0125}$	0.0283 ± 0.0106	0.0275 ± 0.0161	
KS3D	RMSE	2.2190 ± 2.8475	–	0.3768 ± 0.2084	0.2893 ± 0.1132	$\mathbf{0.2211 \pm 0.0310}$	0.2238 ± 0.0440	3
	H^1	2.2237 ± 2.8529	–	0.3784 ± 0.2092	0.2910 ± 0.1143	$\mathbf{0.2222 \pm 0.0313}$	0.2250 ± 0.0446	
	Spectrum	0.1688 ± 0.1718	–	0.0963 ± 0.0129	0.0931 ± 0.0736	0.0859 ± 0.0373	$\mathbf{0.0837 \pm 0.0593}$	
DT3D	RMSE	0.2554 ± 0.2923	–	0.0398 ± 0.0171	0.0358 ± 0.0141	0.0374 ± 0.0147	$\mathbf{0.0304 \pm 0.0091}$	3
	H^1	0.2586 ± 0.2964	–	0.0401 ± 0.0172	0.0361 ± 0.0142	0.0377 ± 0.0148	$\mathbf{0.0306 \pm 0.0091}$	
	Spectrum	0.1554 ± 0.1019	–	0.0673 ± 0.0481	0.0739 ± 0.0452	0.0495 ± 0.0297	$\mathbf{0.0267 \pm 0.0107}$	



(a) 2D tasks. (b) 3D tasks.

Figure 2: In-distribution rollout RMSE averaged across evaluated architectures: four models for 2D tasks and three models for 3D tasks. SaT-0 achieves the lowest cross-model RMSE on six of seven tasks, while SaT-.5 is best on KS3D.

5.1 In-Distribution Benchmark Results

SaT variants dominate long-rollout accuracy. Table 1 reports cross-model aggregate diagnostics on 50-step rollouts for the 6 foundation models across 7 PDE benchmarks; precise definitions of RMSE, the Sobolev H^1 error, and the normalized energy-spectrum error are given in appendix F, and the per-model RMSE breakdown is in appendix M and table 22. Across 24 model–task pairs (12 for 2D, 12 for 3D), a SaT variant gives the best per-model RMSE in 23 cases, with the only exception being MORPH-Ti on DT3D where OPD is slightly lower. The aggregated trends per scenario are even clearer: SaT-0 achieves the lowest cross-model RMSE on six of seven tasks, while SaT-.5 is best on KS3D and differs only marginally from SaT-0 (0.2211 vs. 0.2238; table 1). Relative to SFT, the best SaT variant reduces cross-model RMSE by up to 68.8% on 2D cases (Burgers2D) and up to 90.0% on 3D cases (81.5% on KF3D, 90.0% on KS3D, and 88.1% on DT3D). These results are shown in fig. 2. Thus, SaT improves both the per-architecture best case and the architecture-averaged behavior, which is the relevant criterion for post-training a family of heterogeneous PDE-FMs rather than tuning a single backbone. This ranking is not an artifact of pointwise RMSE alone: the trajectory-level Sobolev H^1 and normalized energy-spectrum of table 1 show a SaT variant best on all 7 in-distribution tasks for both diagnostics, with SaT-0 best on 6 of 7 tasks for each metric. These diagnostics strengthen the interpretation that solver correction improves physically meaningful dynamics, including gradient fidelity and scale-wise energy allocation, rather than merely reducing amplitude error.

Full dynamic loss stabilizes chaotic rollouts. Unrolled training (SaT-1) accounts for most of the improvement over SFT. Replacing its reference targets with solver targets lowers RMSE further in 22 of 24 model–task pairs and on every task's cross-model mean, by 2.4% (Burgers3D) to 22.6% (KS3D). This gap remains when SaT-1 receives as many targets per epoch from a larger static dataset (appendix I) and when the methods are compared at equal wall-clock time (appendix A.2). The relative ranking of SaT-1, SaT-.5, and SaT-0 clarifies the role of the reference anchor in eq. (3). SaT-0 uses the full dynamic loss ($\alpha = 0$): every target is generated by the solver from a student-visited state, so the training signal remains on-policy and physically grounded even after the student has left the reference trajectory. This is exactly the regime where pointwise trajectory matching becomes ill-conditioned: small errors are amplified by chaotic dynamics, and an off-policy reference state can become a poor target for correcting the student's current state. Full dynamic loss therefore dominates the hardest settings, winning 9 of 12 3D model–task pairs. Conversely, the anchor-containing variants remain useful in simpler or more weakly chaotic 2D cases, where the reference trajectory is still informative over the training horizon: SaT-.5 wins 6 of 12 2D model–task pairs. This pattern supports the intended interpretation of α: reference loss can accelerate alignment when the trajectory target remains meaningful, while full dynamic loss is more robust once autoregressive drift and chaotic decorrelation dominate.

SaT rescues temporally under-specified backbones. The strongest architecture-level result is not only that SaT improves average RMSE, but that its gains are largest for backbones whose temporal signal is weakest. DPOT-Ti is pretrained with autoregressive denoising and a multi-frame time aggregator, MPP-B and MORPH-Ti contain explicit temporal attention mechanisms, and Poseidon-T is pretrained with time-conditioned semigroup pairs. In contrast, PDE-S and its 3D extension P3D-S are single-frame transformer backbones, with neither temporal-history input nor temporal-history encoding into structures. This makes them especially vulnerable under SFT, which trains on reference states but never corrects the model's own rollout distribution. SaT directly supplies the missing

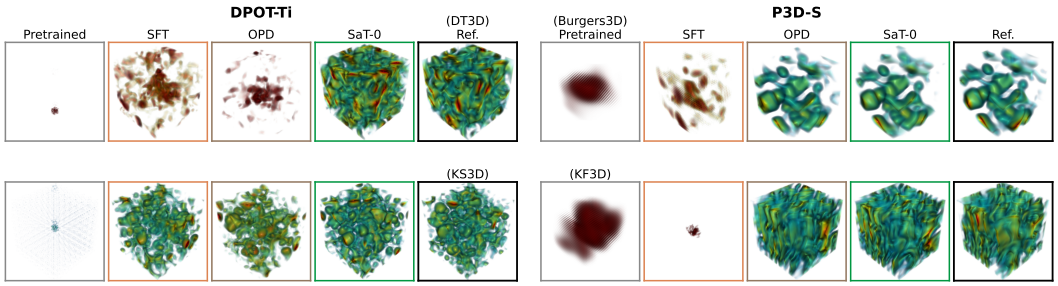


Figure 3: Volumetric fields after 50 autoregressive rollout steps. **Left block**: DPOT-Ti on DT3D and KS3D. **Right block**: P3D-S on Burgers3D and KF3D. SFT collapses into diffuse, sparse, or underdeveloped structures, while OPD partially improves morphology but can still miss the solver reference. SaT-0 recovers coherent volumetric structure and closely matches the reference fields, especially for P3D-S, where SFT is weakest.

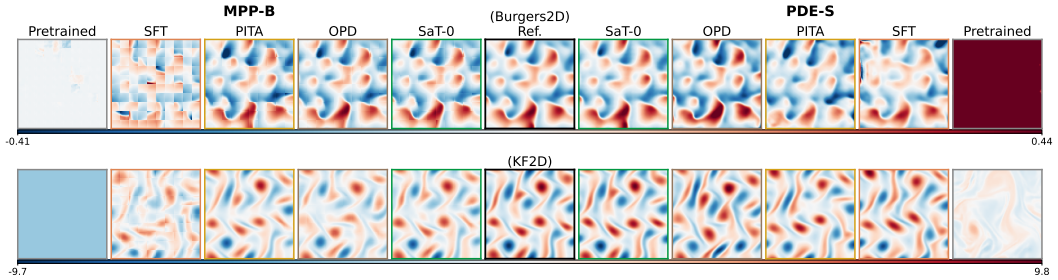
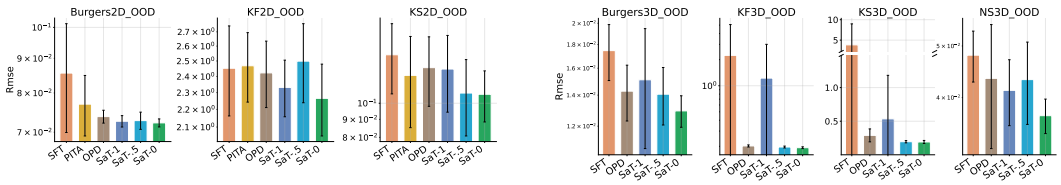


Figure 4: Predicted 2D fields after 50 autoregressive rollout steps; MPP-B (left) and PDE-S (right) on Burgers2D (top) and KF2D (bottom). The solver reference is shown at the center. MPP-B's SFT prediction exhibits pronounced blocky artifacts, arising from the AViT's independent per-patch sparse output projection. SaT-0 removes these artifacts and recovers physically coherent fine-scale structure. PDE-S, whose Swin U-Net decoder fuses multi-scale skip connections, produces smoother SFT fields; SaT further sharpens the prediction toward the reference.

temporal supervision through solver corrections on student-visited states. Under SFT, P3D-S is the weakest 3D model on all four tasks; with SaT-0, it becomes the best model on Burgers3D and DT3D, reducing RMSE up to 97.4% on DT3D (0.6687 to 0.0175). The same improvement shows in harder chaotic tasks: P3D-S improves from 6.2459 to 0.2534 on KS3D with SaT-.5 (95.9%), and from 1.7530 to 0.1478 on KF3D with SaT-0 (91.6%). The 2D sibling PDE-S shows the same pattern, with SaT-0 reducing RMSE up to 73.1% on Burgers2D. The visual comparison in fig. 3 shows the 3D transition: SFT collapses toward diffuse, sparse, or underdeveloped structures, whereas SaT-0 recovers volumetric fields with the correct large-scale morphology. Thus, SaT can supply the missing temporal training signal through the on-policy solver loop itself, allowing even backbones without intrinsic temporal-history encoding to produce stable long rollouts.

On-policy correction removes architecture-specific artifacts. Beyond aggregate metrics, fig. 4 exposes a 2D failure mode that SaT corrects. MPP-B's SFT predictions exhibit a pronounced 10×10 grid of blocky artifacts, an architectural fingerprint of the independent per-patch sparse output projection, which reconstructs each patch without cross-patch convolution or overlap. SFT exacerbates this because the model never trains on its own patchwise artifacts; at inference, small inter-patch inconsistencies are repeatedly fed back and compound across autoregressive steps. SaT corrects the same failure at its source: the on-policy rollout exposes these boundary artifacts to the loss, and dense solver corrections force the next-step map to re-enter a spatially coherent physical field. As shown in fig. 4, SaT-0 eliminates the blocky structure and recovers smooth fields close to the solver reference. In contrast, PDE-S's Swin U-Net decoder already provides overlapping receptive fields through hierarchical skip connections and shifted-window attention, so SFT is smoother; SaT further sharpens the solution toward the reference.



(a) 2D OOD tasks.

(b) 3D OOD tasks.

Figure 5: Zero-shot RMSE on OOD tasks with perturbed physical parameters, averaged across four 2D models or three 3D models. SaT-0 achieves the lowest cross-model RMSE on all 7 OOD tasks.

Solver teachers are more reliable than learned teachers. OPD tests the central limitation discussed in section 1: replacing the numerical solver with a larger PDE-FM also closes the distribution gap, but the teacher remains an approximate neural simulator. This distinction matters in a chaotic rollout. On KF2D, OPD degrades DPOT-Ti relative to SFT from 0.2316 to 0.4018 and PDE-S from 1.0938 to 1.1566. Even when OPD improves over SFT in some cases, the solver teacher remains stronger: on KS3D, the cross-model OPD RMSE is 0.3768, while SaT-.5 reaches 0.2211; on DT3D, OPD gives 0.0398, while SaT-0 gives 0.0304. The reason is structural rather than merely empirical. A learned teacher can provide a dense target, but its target is still produced by a model that may hallucinate missing scales, damp turbulent energy, or inherit pretraining mismatch. The numerical solver provides the PDE evolution at the student's actual state, so its correction remains tied to the governing dynamics even when the student has drifted away from the data trajectory. PITA shows the complementary limitation of residual-only alignment: it improves MPP-B on KF2D (1.4343 to 0.8410) but degrades DPOT-Ti on the same task (0.2316 to 0.3674), and on Burgers2D it improves DPOT-Ti (0.0126 to 0.0103) but degrades PDE-S (0.0446 to 0.0515). Soft physics constraints are therefore useful for some architecture–PDE pairs, but less consistent than explicit solver targets. Beyond accuracy, the solver teacher is also substantially cheaper: on KS3D, SaT-0 completes training in $\sim$3.1–5.9 h depending on the backbone, while OPD requires $\sim$18.7–22.0 h for the student training alone plus $\sim$14.5 h of teacher preparation, for a total cost of $\sim$33–37 h, or about 6.2–$10.9\times$ the SaT-0 wall-clock time (appendix A.1 and table 6).

5.2 Out-of-Distribution Zero-Shot Results

Table 2: OOD zero-shot RMSE (mean $\pm$ std) for 2D and 3D tasks with perturbed physical parameters (see appendix E.4). Best per task **bolded**. PITA does not support 3D tasks (–).

Task	SFT	PITA	OPD	SaT-1	SaT-.5	SaT-0	N models
Burgers2D-OOD	0.0856 ± 0.0157	0.0770 ± 0.0079	0.0738 ± 0.0016	0.0727 ± 0.0014	0.0728 ± 0.0021	$\mathbf{0.0723 \pm 0.0010}$	4
KS2D-OOD	0.1373 ± 0.0310	0.1200 ± 0.0347	0.1262 ± 0.0282	0.1251 ± 0.0307	0.1069 ± 0.0261	$\mathbf{0.1060 \pm 0.0175}$	4
KF2D-OOD	2.4524 ± 0.2899	2.4685 ± 0.2256	2.4226 ± 0.2118	2.3316 ± 0.1738	2.4989 ± 0.2597	$\mathbf{2.2652 \pm 0.2140}$	4
Burgers3D-OOD	0.0175 ± 0.0024	–	0.0143 ± 0.0020	0.0131 ± 0.0012	0.0141 ± 0.0020	$\mathbf{0.0129 \pm 0.0010}$	3
KF3D-OOD	1.0658 ± 1.1758	–	0.2021 ± 0.0056	0.1988 ± 0.0042	0.1963 ± 0.0049	$\mathbf{0.1929 \pm 0.0045}$	3
KS3D-OOD	2.6798 ± 3.5134	–	0.2897 ± 0.0961	0.2004 ± 0.0188	0.1956 ± 0.0168	$\mathbf{0.1920 \pm 0.0226}$	3
DT3D-OOD	0.2487 ± 0.2838	–	0.0435 ± 0.0114	0.0425 ± 0.0063	0.0432 ± 0.0076	$\mathbf{0.0370 \pm 0.0028}$	3

Full dynamic loss generalizes to shifted physics. Table 2 and fig. 5 report zero-shot RMSE under physical-parameter shifts: stronger nonlinear coupling for KS, stronger turbulence for KF, higher viscosity for DT3D, and weakened diffusion and convection for Burgers. SaT-0 gives the lowest cross-model RMSE on all seven OOD tasks, with the largest gains in 3D: 81.9% on KF3D-OOD, 92.8% on KS3D-OOD, and 85.1% on DT3D-OOD relative to SFT. This supports the main mechanism rather than task-specific fitting: the student is trained on its own rollout states, including states that have drifted from the reference data, and the solver provides dynamically consistent supervision at each of them. The appendix diagnostics lead to the same conclusion beyond RMSE: table 16 shows SaT-0 best on all seven OOD H^1 tasks, and a SaT variant best on six of seven normalized spectrum-error tasks. Per-model OOD results are reported in appendix N and table 23.

6 Concluding Remarks

We introduced **Solver-as-Teacher (SaT)**, an on-policy post-training framework that uses a numerical solver as a dense teacher for the states visited by a PDE-FM during its own autoregressive rollout.

This changes post-training from offline trajectory fitting into solver-guided correction of the model's induced dynamics. Across seven 2D and 3D PDE benchmarks and six pretrained backbones, the experiments show that SaT variants achieve the best 50-step rollout RMSE in 23 of 24 in-distribution model–task pairs, reduce cross-model RMSE over SFT by up to 90.0%, and give the lowest cross-model RMSE on all seven OOD parameter-shift benchmarks. The method comparisons further support the mechanism: full dynamic loss is strongest across most 3D cases and all OOD aggregate settings, solver teachers are more reliable than learned FM teachers in our comparisons, and the gains are largest for backbones whose temporal signal is weakest under standard SFT.

Limitations. One limiting area is that SaT by construction relies on access to reliable numerical solvers at training time. Its targets are solver-derived references. They are accurate only up to the governing equation and its discretization, and SaT requires a teacher whose dynamics remain valid at the states the student visits. A teacher coarsened until it diverges or changes the dynamics yields a correspondingly worse student: on KS2D, coarsening the teacher from 128^2 to 80^2, where it diverges after 36 steps, raises the student's cross-model RMSE from 0.090 to 0.268. We have not tested governing equations that are known to be incomplete or misspecified; pairing an inexpensive coarse solver with a learned closure is a natural direction. Also, our evaluation does not yet extend to complex geometries or unstructured meshes. Finally, SaT's efficiency is closely tied to successful solver parallelization. This poses an interesting direction for asynchronous training strategies.

Despite this, the presented results already demonstrate the promise of solver-guided, on-policy post-training for improving the robustness and long-horizon fidelity of PDE-FMs, and may serve as a key building block for future scientific foundation model pipelines.

References

[1] Benjamin Holzschuh, Qiang Liu, Georg Kohl, and Nils Thuerey. Pde-transformer: Efficient and versatile transformers for physics simulations. *arXiv preprint arXiv:2505.24717*, 2025.

[2] 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.

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

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

[5] Michael McCabe, Payel Mukhopadhyay, Tanya Marwah, Bruno Regaldo-Saint Blancard, Francois Rozet, Cristiana Diaconu, Lucas Meyer, Kaze WK Wong, Hadi Sotoudeh, Alberto Bietti, et al. Walrus: A cross-domain foundation model for continuum dynamics. *arXiv preprint arXiv:2511.15684*, 2025.

[6] Rui Zhang, Qi Meng, Han Wan, Yang Liu, Zhi-Ming Ma, and Hao Sun. Omnifluids: Physics pre-trained modeling of fluid dynamics. *arXiv preprint arXiv:2506.10862*, 2025.

[7] Mahindra Singh Rautela, Alexander Most, Siddharth Mansingh, Bradley C Love, Alexander Scheinker, Diane Oyen, Nathan Debardeleben, Earl Lawrence, and Ayan Biswas. Morph: Pde foundation models with arbitrary data modality. *arXiv preprint arXiv:2509.21670*, 2025.

[8] Benjamin Holzschuh, Georg Kohl, Florian Redinger, and Nils Thuerey. P3d: Scalable neural surrogates for high-resolution 3d physics simulations with global context. *arXiv preprint arXiv:2509.10186*, 2025.

[9] Jeffrey Lai, Anthony Bao, and William Gilpin. Panda: A pretrained forecast model for chaotic dynamics. *arXiv preprint arXiv:2505.13755*, 2025.

[10] Armand Kassaï Koupaï, Jorge Mifsut Benet, Yuan Yin, Jean-Noël Vittaut, and Patrick Gallinari. Geps: Boosting generalization in parametric pde neural solvers through adaptive conditioning. In *Advances in Neural Information Processing Systems*, 2024.

[11] Aman Gupta, Aditi Sheshadri, Sujit Roy, Johannes Schmude, Vishal Gaur, Wei Ji Leong, Manil Maskey, and Rahul Ramachandran. Finetuning ai foundation models to develop subgrid-scale parameterizations: A case study on atmospheric gravity waves. *Journal of Advances in Modeling Earth Systems*, 17(11):e2025MS005075, 2025.

[12] 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.

[13] Zongyi Li, Miguel Liu-Schiaffini, Nikola Kovachki, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Learning dissipative dynamics in chaotic systems. *arXiv preprint arXiv:2106.06898*, 2021.

[14] Yi He, Yiming Yang, Xiaoyuan Cheng, Hai Wang, Xiao Xue, Boli Chen, and Yukun Hu. Chaos meets attention: Transformers for large-scale dynamical prediction. *arXiv preprint arXiv:2504.20858*, 2025.

[15] Dibyajyoti Nayak and Somdatta Goswami. Ti-deeponet: Learnable time integration for stable long-term extrapolation. *Computer Methods in Applied Mechanics and Engineering*, 456:118960, 2026.

[16] Lennon Shikhman. Forcing and diagnosing failure modes of fourier neural operators across diverse pde families. *arXiv preprint arXiv:2601.11428*, 2026.

[17] Xintong Zou, Zhijie Li, Yunpeng Wang, Huiyu Yang, and Jianchun Wang. Uncertainty quantification and stability of neural operators for prediction of three-dimensional turbulence. *Journal of Computational Physics*, page 114640, 2025.

[18] Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668. JMLR Workshop and Conference Proceedings, 2010.

[19] Jonas Mikhaeil, Zahra Monfared, and Daniel Durstewitz. On the difficulty of learning chaotic dynamics with rnns. *Advances in neural information processing systems*, 35:11297–11312, 2022.

[20] Ruoxi Jiang, Peter Y. Lu, Elena Orlova, and Rebecca Willett. Training neural operators to preserve invariant measures of chaotic attractors. In *Advances in Neural Information Processing Systems*, 2023.

[21] Hao Wei, Aleksandra Franz, Bjoern Malte List, and Nils Thuerey. Inc: An indirect neural corrector for auto-regressive hybrid pde solvers. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[22] Guiyao Tie, Zeli Zhao, Dingjie Song, Fuyang Wei, Rong Zhou, Yurou Dai, Wen Yin, Zhejian Yang, Jiangyue Yan, Yao Su, et al. A survey on post-training of large language models. *arXiv preprint arXiv:2503.06072*, 2025.

[23] Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *The twelfth international conference on learning representations*, 2024.

[24] Arnav Gudibande, Eric Wallace, Charlie Snell, Xinyang Geng, Hao Liu, Pieter Abbeel, Sergey Levine, and Dawn Song. The false promise of imitating proprietary llms. *arXiv preprint arXiv:2305.15717*, 2023.

[25] Yaxuan Li, Yuxin Zuo, Bingxiang He, Jinqian Zhang, Chaojun Xiao, Cheng Qian, Tianyu Yu, Huan-ang Gao, Wenkai Yang, Zhiyuan Liu, et al. Rethinking on-policy distillation of large language models: Phenomenology, mechanism, and recipe. *arXiv preprint arXiv:2604.13016*, 2026.

[26] Congcong Zhu, Xiaoyan Xu, Jiayue Han, and Jingrun Chen. Physics-informed temporal alignment for auto-regressive pde foundation models. *arXiv preprint arXiv:2505.10930*, 2025.

[27] Shashank Subramanian, Peter Harrington, Kurt Keutzer, Wahid Bhimji, Dmitriy Morozov, Michael W Mahoney, and Amir Gholami. Towards foundation models for scientific machine learning: Characterizing scaling and transfer behavior. *Advances in Neural Information Processing Systems*, 36:71242–71262, 2023.

[28] Amin Totounferoush, Serge Kotchourko, Michael W Mahoney, and Steffen Staab. Paving the way for scientific foundation models: enhancing generalization and robustness in pdes with constraint-aware pre-training. *arXiv preprint arXiv:2503.19081*, 2025.

[29] Louis Serrano, Armand Kassaï Koupaï, Thomas X Wang, Pierre Erbacher, and Patrick Gallinari. Zebra: In-context generative pretraining for solving parametric pdes. *arXiv preprint arXiv:2410.03437*, 2024.

[30] Yadi Cao, Yuxuan Liu, Liu Yang, Rose Yu, Hayden Schaeffer, and Stanley Osher. Vicon: Vision in-context operator networks for multi-physics fluid dynamics prediction. *arXiv preprint arXiv:2411.16063*, 2024.

[31] Liu Yang, Siting Liu, Tingwei Meng, and Stanley J Osher. In-context operator learning with data prompts for differential equation problems. *Proceedings of the National Academy of Sciences*, 120(39):e2310142120, 2023.

[32] Hao Wei, Björn List, and Nils Thuerey. ReViT: Rotational-equivariant vision transformers for neural PDE solvers. In *Forty-third International Conference on Machine Learning*, 2026.

[33] Johannes Brandstetter, Daniel Worrall, and Max Welling. Message passing neural pde solvers. *arXiv preprint arXiv:2202.03376*, 2022.

[34] Kiwon Um, Robert Brand, Yun Raymond 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, 2020.

[35] Xinquan Huang and Paris Perdikaris. Physicscorrect: A training-free approach for stable neural pde simulations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 22057–22065, 2026.

[36] Yair Schiff, Zhong Yi Wan, Jeffrey B. Parker, Stephan Hoyer, Volodymyr Kuleshov, Fei Sha, and Leonardo Zepeda-Nunez. Dyslim: Dynamics stable learning by invariant measure for chaotic systems. *arXiv preprint arXiv:2402.04467*, 2024.

[37] Kaifeng Bi, Lingxi Xie, Hengheng Zhang, Xin Chen, Xiaotao Gu, and Qi Tian. Accurate medium-range global weather forecasting with 3d neural networks. *Nature*, 619(7970):533–538, 2023.

[38] Kim Stachenfeld, Drummond Buschman Fielding, Dmitrii Kochkov, Miles Cranmer, Tobias Pfaff, Jonathan Godwin, Can Cui, Shirley Ho, Peter Battaglia, and Alvaro Sanchez-Gonzalez. Learned simulators for turbulence. In *International conference on learning representations*, 2022.

[39] Zhiqing Sun, Yiming Yang, and Shinjae Yoo. A neural pde solver with temporal stencil modeling. In *International Conference on Machine Learning*, pages 33135–33155. PMLR, 2023.

[40] Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021.

[41] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[42] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[43] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

[44] Hangwei Zhang, Chun Kang, Yan Wang, and Difan Zou. F-adapter: Frequency-adaptive parameter-efficient fine-tuning in scientific machine learning. *arXiv preprint arXiv:2509.23173*, 2025.

[45] Shuhao Cao, Francesco Brarda, Ruipeng Li, and Yuanzhe Xi. Spectral-refiner: Accurate fine-tuning of spatiotemporal fourier neural operator for turbulent flows. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

[46] Siddharth Mansingh, James Amarel, Ragib Arnab, Arvind Mohan, Kamaljeet Singh, Gerd J Kunde, Nicolas Hengartner, Benjamin Migliori, Emily Casleton, Nathan A Debardeleben, et al. Towards reasoning for pde foundation models: A reward-model-driven inference-time-scaling algorithm. *arXiv preprint arXiv:2509.02846*, 2025.

[47] Felix Koehler, Simon Niedermayr, Rüdiger Westermann, and Nils Thuerey. Apebench: A benchmark for autoregressive neural emulators of pdes. *Advances in Neural Information Processing Systems*, 37:120252–120310, 2024.

[48] Aleksandra Franz, Hao Wei, Luca Guastoni, and Nils Thuerey. Pict–a differentiable, gpu-accelerated multi-block piso solver for simulation-coupled learning tasks in fluid dynamics. *Journal of Computational Physics*, page 114433, 2025.

[49] Bjoern List, Li-Wei Chen, Kartik Bali, and Nils Thuerey. Differentiability in unrolled training of neural physics simulators on transient dynamics. *Computer Methods in Applied Mechanics and Engineering*, 433:117441, 2025.

[50] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[51] 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.

[52] Jannis Becktepe, Aleksandra Franz, Nils Thuerey, and Sebastian Peitz. Plug-and-play benchmarking of reinforcement learning algorithms for large-scale flow control. In *Forty-third International Conference on Machine Learning*, 2026.

Appendix

A Training Method Computational Efficiency

We profile the per-iteration computational cost of three training paradigms: Supervised Fine-Tuning (SFT), Solver-as-Teacher (SaT), and On-Policy Distillation with an FM teacher (OPD). All methods share the same student model (PDE-S, 33M parameters) and are evaluated on the KF2D task (160^2 resolution, batch size $B = 4$, single NVIDIA L40S GPU). Each training step is decomposed into four timed components using asynchronous CUDA events: (1) model forward (student rollout), (2) solver/teacher (target generation), (3) backward (gradient computation, including re-forward for DAgger), and (4) optimizer (gradient clipping and AdamW update). We report the mean over 20 timed iterations after 5 warmup iterations.

SFT calibration. For fair comparison, SFT unroll steps S are set to match two reference points per SaT configuration: (i) the student rollout horizon $S = K$, and (ii) the total solver compositions $N_{\text{solver}}(K, m_{\max}) = \sum_{k=0}^{K-1} \min(m_{\max}, K - k)$. With $K \in \{1, 5\}$ and $m_{\max} = 5$, this yields $S \in \{1, 5, 15\}$.

Gradient modes. Both off-policy rollout training and on-policy training generate a K-step autoregressive student rollout $\hat{u}_0 \xrightarrow{\mathcal{G}_\theta} \hat{u}_1 \xrightarrow{\mathcal{G}_\theta} \cdots \xrightarrow{\mathcal{G}_\theta} \hat{u}_K$, and the gradient computation through this chain admits two modes. Because the present study uses rollout horizons up to $K = 5$, BPTT remains feasible; however, DAgger/detach provides a more scalable option for larger K or memory-constrained settings. We therefore benchmark both modes to quantify the time–memory trade-off, while leaving a broader comparison of long-horizon gradient estimators to prior work [49].

Backpropagation through time (BPTT). The full rollout chain retains the computational graph, and a single backward call computes depth-K Jacobian-vector products. This provides inter-step gradient signal—a parameter change at step 0 affects the loss at step K—but activation memory scales as $O(K)$ because all intermediate states must be kept live.

DAgger/detach. Inspired by the DAgger framework for imitation learning [50], this mode detaches the autoregressive chain: each rollout state is generated without gradient, then independently re-forwarded to compute depth-1 gradients via K separate backward calls. The SFT analogue ("detach" in Tables 3–4) applies the same per-step detach-and-reforward pattern to the supervised unroll. Because no intermediate activations are retained across steps, activation memory is $O(1)$ regardless of the rollout horizon, at the cost of losing the inter-step gradient signal. This trade-off is primarily motivated by memory constraints when scaling to large foundation models or rolling out long horizons, as DAgger/detach keeps the memory footprint bounded and enables training configurations that would otherwise be infeasible.

Table 3: Per-iteration component breakdown (ms) under BPTT gradient mode. SaT uses the Exponax pseudo-spectral solver; OPD uses PDE-L (518M params) as teacher. Measurements use $B = 4$ at 160^2 resolution on an NVIDIA L40S GPU.

Method	Forward	Solver/Teacher	Backward	Optimizer	**Total**
SFT ($S = 1$)	18.6	—	19.9	1.9	**40.8**
SFT ($S = 5$)	93.8	—	75.1	2.4	**172.3**
SFT ($S = 15$)	270.2	—	228.5	2.8	**503.9**
SaT ($K = 1, m = 5$)	20.4	4.4	24.0	2.0	**50.9**
SaT ($K = 5, m = 5$)	90.9	41.5	75.3	2.3	**210.1**
OPD ($K = 1, m = 5$)	18.8	52.2	0.3	1.0	**72.4**
OPD ($K = 5, m = 5$)	92.0	708.5	2.3	1.5	**804.5**

SaT adds modest overhead; OPD is teacher-dominated. At $K = 5$, SaT (BPTT) requires 210 ms per step — the numerical solver accounts for only 20% of the total cost (42 ms). The remaining 80% is the student's own forward and backward passes, which scale linearly with K. In contrast, OPD at $K = 5$ requires 805 ms, with the FM teacher consuming 88% of the step cost (709 ms). The teacher

Table 4: Per-iteration component breakdown (ms) under DAgger/detach gradient mode. The backward column includes the re-forward passes (depth-1 each) that replace the BPTT chain. Same setup as table 3.

Method	Forward	Solver/Teacher	Backward	Optimizer	**Total**
SFT ($S = 1$, detach)	18.9	—	21.0	1.9	**42.1**
SFT ($S = 5$, detach)	97.4	—	106.2	2.2	**208.8**
SFT ($S = 15$, detach)	296.8	—	321.2	2.2	**630.3**
SaT ($K = 1$, DAgger)	17.8	4.5	42.9	1.9	**67.1**
SaT ($K = 5$, DAgger)	77.8	40.3	202.0	2.0	**322.1**
OPD ($K = 1$, DAgger)	17.0	51.6	17.0	0.9	**86.6**
OPD ($K = 5$, DAgger)	74.1	712.8	91.7	1.3	**879.9**

is approximately $17\times$ more expensive than the numerical solver at the same horizon, reflecting the $15.6\times$ parameter ratio between PDE-L (518M) and the lightweight Exponax integrator.

Matched-horizon SFT comparison. When S is calibrated to the total solver compositions N_{solver}, SFT provides a compute-matched baseline. At $S = 15$ (matching SaT $K = 5$, $m_{\max} = 5$), SFT (BPTT) takes 504 ms — $2.4\times$ more than SaT's 210 ms for the equivalent number of model evaluations. This gap arises because SaT's solver branches (42 ms) are substantially cheaper than the additional depth-15 backward pass through the full unroll chain that SFT requires.

BPTT vs. DAgger trade-off. Table 5 compares the two gradient modes. DAgger is consistently slower due to (i) K separate backward calls vs. one, each incurring Python dispatch and gradient-accumulation overhead, and (ii) additional depth-1 re-forward passes. The penalty is most pronounced for SaT ($+53\%$), where the re-forward cost dominates. For OPD the overhead is modest ($+9\%$) because the teacher's forward pass — unchanged between modes — accounts for the vast majority of step time. In practice, DAgger remains preferable for large K due to its $O(1)$ activation memory, enabling training configurations that would otherwise exhaust GPU memory.

Table 5: BPTT vs. DAgger per-step wall-clock time at $K = 5$, $m_{\max} = 5$ (or $S = 15$ for SFT). DAgger trades wall-clock overhead for $O(1)$ activation memory. Same setup as Tables 3–4.

Config	BPTT (ms)	DAgger (ms)	Overhead
SFT ($S = 15$)	503.9	630.3	$+25\%$
SaT ($K = 5$, $m = 5$)	210.1	322.1	$+53\%$
OPD ($K = 5$, $m = 5$)	804.5	879.9	$+9\%$

A.1 End-to-End Training Wall-Clock Time

The per-iteration breakdowns in Tables 3–4 characterize *compute cost per step*; here we report the aggregate wall-clock time observed for full training runs on the KS3D benchmark (64^3, single NVIDIA A100 GPU). Table 6 lists the approximate total training time for each model–method pair, including both the SFT warm-up phase and the main training phase. SFT is the cheapest method for every backbone, finishing in $\sim$1.4–2.6 h depending on the model. SaT variants add modest overhead: SaT-0, the best-performing variant, ranges from $\sim$3.1 h (MORPH-Ti) to $\sim$5.9 h (P3D-S), while SaT-1 and SaT-.5 fall in the same ballpark ($\sim$2.4–6.1 h). In contrast, OPD requires $\sim$18.7–22.0 h per run for the student training alone, roughly 3.7–$6.2\times$ the cost of SaT-0, because every on-policy step queries the frozen P3D-L teacher. Crucially, these OPD times exclude teacher preparation: the P3D-L teacher itself requires $\sim$14.5 h of fine-tuning before it can serve as the OPD oracle, bringing the true end-to-end OPD cost to $\sim$33–37 h, or about 6.2–$10.9\times$ SaT-0.

A.2 Accuracy at Equal Wall-Clock Time and Equal FLOPs

To compare methods at matched training budgets, we retrained SFT, SaT-1, SaT-.5, SaT-0 and OPD from scratch on three tasks with two backbones each (DPOT-Ti and PDE-S on KS2D; DPOT-Ti and

Table 6: End-to-end training wall-clock time for all model–method pairs on KS3D (64^3, single NVIDIA A100 GPU). All times are approximate.

Model	Method	Total Time
DPOT-Ti	SFT	$\sim$1.5 h
DPOT-Ti	OPD	$\sim$18.7 h
DPOT-Ti	SaT-1	$\sim$2.4 h
DPOT-Ti	SaT-.5	$\sim$2.9 h
DPOT-Ti	SaT-0	$\sim$3.6 h
MORPH-Ti	SFT	$\sim$1.4 h
MORPH-Ti	OPD	$\sim$19.2 h
MORPH-Ti	SaT-1	$\sim$2.8 h
MORPH-Ti	SaT-.5	$\sim$3.0 h
MORPH-Ti	SaT-0	$\sim$3.1 h
P3D-S	SFT	$\sim$2.6 h
P3D-S	OPD	$\sim$22.0 h
P3D-S	SaT-1	$\sim$5.6 h
P3D-S	SaT-.5	$\sim$6.1 h
P3D-S	SaT-0	$\sim$5.9 h
OPD teacher preparation		
P3D-L	–	$\sim$14.5 h

P3D-S on Burgers3D and KS3D). Each method uses its own recipe from table 22, with one seed, and all runs use NVIDIA A40 GPUs. Because these are separate runs, their absolute RMSE differs from table 22, so we report comparisons within each cell. Each entry gives the best method at that budget and its relative RMSE change: for SaT-0 and SaT-.5 relative to SaT-1, and for SaT-1 relative to SaT-0, so a positive value is an on-policy improvement. Each point uses the lowest test RMSE reached within the budget.

Table 7: Best method and relative RMSE change at equal wall-clock time. Positive values are improvements of the on-policy variant over SaT-1.

Cell	1 h	3 h	6 h	9 h
Burgers3D / DPOT-Ti	SaT-0 $+3\%$	SaT-0 $+27\%$	SaT-0 $+28\%$	SaT-0 $+28\%$
Burgers3D / P3D-S	not converged	SaT-1 -18%	SaT-1 -1%	SaT-0 $+5\%$
KS2D / DPOT-Ti	SaT-.5 $+39\%$	SaT-.5 $+33\%$	SaT-.5 $+33\%$	SaT-.5 $+33\%$
KS2D / PDE-S	SaT-0 0%	SaT-0 $+11\%$	SaT-0 $+15\%$	SaT-0 $+16\%$
KS3D / DPOT-Ti	SaT-0 $+20\%$	SaT-0 $+14\%$	SaT-0 $+14\%$	SaT-0 $+20\%$
KS3D / P3D-S	not converged	SaT-.5 $+42\%$	SaT-.5 $+26\%$	SaT-.5 $+26\%$

An on-policy variant outperforms SaT-1 in five of six settings after three and six hours, and in all six after nine hours. When SaT-0 reaches the converged accuracy of SaT-1, it does so with a median $2.6\times$ less wall-clock time.

For FLOPs, we count the student's forward pass from the training graph and one solver macro-step from the compiled solver's XLA cost model. At $K = m = 5$, a training step makes 15 teacher calls and costs 15 student forward-pass equivalents: five forward steps plus a backward pass of about ten. At 160^2, a solver macro-step costs 0.11 GFLOP against 3.7–25 GFLOP for one student forward pass, an overhead below 3%. At 64^3, the costs are 5.3 and 13 GFLOP, and the overhead rises to $1.19\times$–$1.41\times$. Table 8 expresses budgets as fractions of the end-to-end cost of the SaT-1 recipe and uses the same convention as table 7.

Charging the solver for every FLOP does not change the outcome: an on-policy variant leads in all four settings from a quarter of the SaT-1 budget onward, with a median improvement of 37% at that budget and 20% at the full budget.

Table 8: Solver FLOPs overhead and best method at equal training FLOPs, as a fraction of the SaT-1 budget.

Cell	Overhead	$0.1\times$	$0.25\times$	$0.5\times$	$1.0\times$
Burgers3D / DPOT-Ti	$1.41\times$	SaT-1 -7%	SaT-0 $+19\%$	SaT-0 $+28\%$	SaT-0 $+28\%$
Burgers3D / P3D-S	$1.19\times$	not converged	SaT-0 $+64\%$	SaT-0 $+4\%$	SaT-0 $+0\%$
KS3D / DPOT-Ti	$1.41\times$	SaT-1 -1%	SaT-0 $+14\%$	SaT-0 $+13\%$	SaT-0 $+14\%$
KS3D / P3D-S	$1.19\times$	not converged	SaT-.5 $+54\%$	SaT-.5 $+26\%$	SaT-.5 $+26\%$

A.3 Inference Speed of the Surrogate and the Solver

Table 9 reports the time to advance one dataset frame at batch size 1 on one NVIDIA A40 in fp32 (median of 30 timed repetitions after 5 warm-up repetitions). The surrogate column is the fastest student for that task. Timing depends only on the architecture, so the students are timed with randomly initialized weights.

Table 9: Time per frame at batch size 1 (ms). The last column states which side is faster and by how much.

Task	Sub-steps	Solver	Surrogate	Faster
KS2D (160^2)	1	0.195	7.09	solver, $36.4\times$
Burgers2D (160^2)	1	0.197	7.14	solver, $36.3\times$
KF2D (160^2)	20	2.55	7.06	solver, $2.8\times$
KS3D (64^3)	1	0.368	7.94	solver, $21.5\times$
Burgers3D (64^3)	1	0.375	8.50	solver, $22.7\times$
KF3D (64^3)	20	8.62	8.84	tie, $1.0\times$
DT3D (64^3)	20	8.22	8.86	tie, $1.1\times$
KS3D (96^3)	1	1.22	14.76	solver, $12.1\times$
KF3D (96^3)	20	44.50	14.75	surrogate, $3.0\times$
TCF3D (PICT)	1	23.34	8.94	surrogate, $2.6\times$

For a lightweight pseudo-spectral solver with few sub-steps, the solver is faster. The surrogate becomes faster when a frame is expensive: many sub-steps on a fine grid (KF3D at 96^3, $3.0\times$) or an implicit iterative solve (PICT, $2.6\times$).

B Solver–Model Bridge Efficiency

A key engineering challenge in the Solver-as-Teacher framework is the *bridge* between the student FM (PyTorch) and the teacher solver (JAX/Exponax). During on-policy training, every rollout step requires transferring states from PyTorch to JAX for solver evaluation and transferring the resulting targets back. With multi-step lookahead depth $m_{\max}$, each training iteration involves $\mathcal{O}(K \cdot m_{\max})$ such round-trips, making bridge overhead a potential bottleneck. We implement and benchmark three bridge strategies:

1. **DLPack zero-copy** (`gpu_dlpack`): Both JAX and PyTorch allocate on the same GPU. Tensors are exchanged via the DLPack protocol (`jnp.from_dlpack` / `torch.from_dlpack`), which shares the underlying device pointer without any data movement.
2. **CPU bridge** (`gpu_cpu_bridge`): The solver runs on GPU via JAX, but data is transferred through host memory: `Tensor.cpu().numpy()` $\rightarrow$ `jnp.array()` on the forward path and `np.array()` $\rightarrow$ `torch.from_numpy().cuda()` on the return path. This incurs two GPU$\leftrightarrow$CPU copies per round-trip.
3. **CPU solver** (`cpu_solver`): JAX runs entirely on CPU (`JAX_PLATFORMS=cpu`). This avoids GPU memory contention but is dramatically slower for spectral solvers, which are compute-heavy.

We benchmark the full depth-first multi-step branching loop, which mirrored the actual training pipeline, on Kolmogorov Flow (160×160, $\mathrm{Re} \approx 100$) with batch size $B = 4$, rollout horizons

$K \in \{3, 5\}$, lookahead depths $m_{\max} \in \{1, 3\}$, and solver precisions `float32` / `float64`. All timings are averaged over 50 iterations on an NVIDIA L40S GPU. The DLPack bridge reduces bridge overhead by 5–10× compared to the CPU bridge, yielding 1.10–1.39× end-to-end speedup across configurations. The CPU solver is 50–77× slower than either GPU mode, confirming that GPU-resident spectral solvers are essential for practical training throughput. Incremental GPU memory above the fixed CUDA context (427 MB) is ∼148 MB (`float32`) / ∼178 MB (`float64`) for DLPack, comparable to the CPU bridge, while the CPU solver uses only ∼93 MB since JAX allocates on host. Figure 6 visualizes these results.

Table 10: Total iteration time (ms) and incremental GPU memory (above the fixed 427 MB CUDA context) for each bridge mode and precision, reported as mean ± std across four $(K, m_{\max})$ configurations ($K \in \{3, 5\}$, $m_{\max} \in \{1, 3\}$). $B = 4$, 160×160, NVIDIA L40S.

	Iter Time (ms)		Incr. Memory (MB)	
Mode	`float32`	`float64`	`float32`	`float64`
DLPack zero-copy	21.8 ± 7.6	40.2 ± 17.3	148 ± 1	178 ± 1
CPU bridge	28.4 ± 11.0	43.8 ± 17.9	164 ± 16	177 ± 1
CPU solver	1624.2 ± 856.2	2624.1 ± 1482.0	93 ± 1	93 ± 1

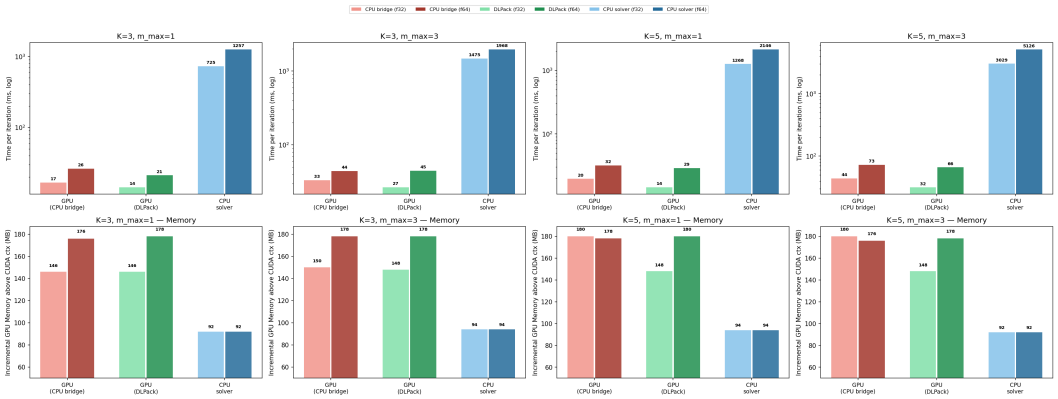


Figure 6: JAX↔PyTorch bridge benchmark. **Top row**: per-iteration wall-clock time (log scale) for three bridge strategies across four $(K, m_{\max})$ configurations, with `float32` (lighter) and `float64` (darker) bars. **Bottom row**: incremental GPU memory above the fixed CUDA context (427 MB) for each mode. DLPack zero-copy consistently achieves the lowest latency while maintaining comparable memory footprint to the CPU bridge.

C Solver Backend Vectorization

The multi-step solver branching in SaT (Eq. 2) requires $\sum_{k=0}^{K-1} m_k$ solver evaluations per training step, each applied to a mini-batch of B samples. The total number of individual solver macro-steps is $B \times \sum m_k$, which grows as $O(B \cdot K \cdot m_{\max})$. For Kolmogorov Flow at 160×160 resolution with 20 ETDRK2 sub-steps per macro-step, the solver branching accounts for 80%+ of per-iteration wall-clock time when executed naively. We implement and benchmark three solver backends that address two root bottlenecks: (1) sequential per-sample solver calls (Python-level loop over the batch dimension) and (2) suboptimal scheduling of the branch–depth computation graph.

The three backends represent a *no-vectorization baseline* and two vectorization strategies that differ in how the branch–depth computation graph is traversed:

1. **No vectorization** — iterates by depth level j; at each depth, all active branches are concatenated into a mega-batch, but each sample is solved individually via a sequential Python loop. This baseline isolates the raw solver cost without any hardware-level parallelism. Because both depth-first and branch-first traversal orders execute the same total number of single-sample solver calls ($B \times \sum m_k$), they yield identical wall-clock times; we therefore report a single baseline.

2. **State vectorization** — same depth-first scheduling, but the mega-batch at each depth level is processed in a single vectorized solver dispatch (via `jax.vmap`), eliminating the per-sample loop. At depth j, the batch size equals B times the number of branches still active, maximizing GPU occupancy.
3. **Solver vectorization** — iterates by branch k; each branch's full chain $[\mathcal{S}^1(\hat{u}_k), \ldots, \mathcal{S}^{m_k}(\hat{u}_k)]$ is computed in a single fused dispatch (`jax.vmap` + `jax.lax.scan`), vectorizing across all B samples and chaining through m_k steps without returning to the host framework. This issues K sequential XLA dispatches (one per branch), each with batch size B.

Setup. We benchmark on 2D Kolmogorov Flow (160×160, $\mathrm{Re} \approx 100$) using the Exponax pseudo-spectral solver with 20 ETDRK2 sub-steps per macro-step. Batch size $B = 4$, rollout horizon $K = 10$, truncate strategy, NVIDIA L40S GPU. All timings are averaged over 10 iterations after 2 warmup iterations. Numerical equivalence (max absolute difference $= 0$) is verified across all backends.

Table 11: Wall-clock time (s) and speedup relative to the no-vectorization baseline for three solver backends across two lookahead depths. $K = 10$, $B = 4$, 160×160, truncate strategy, NVIDIA L40S.

	$m_{\max} = 5$ (40 steps)		$m_{\max} = 8$ (52 steps)	
Backend	Time (s)	Speedup	Time (s)	Speedup
No vectorization	0.464	1.00×	0.596	1.00×
State vectorization	0.033	14.20×	0.047	12.74×
Solver vectorization	0.102	4.55×	0.126	4.73×

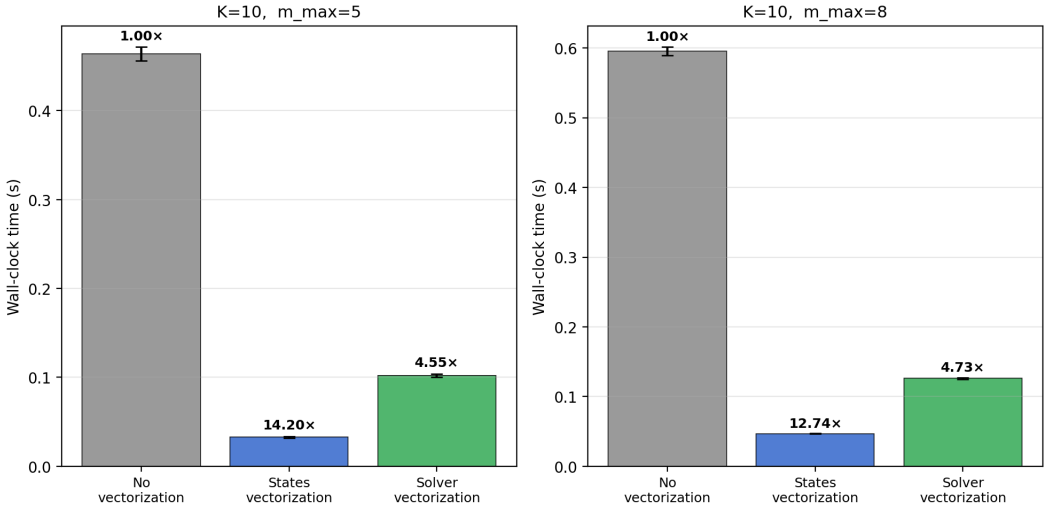


Figure 7: Wall-clock comparison of three solver backends on Kolmogorov Flow (160×160, $B = 4$, GPU). Speedup factors are annotated above each bar relative to the no-vectorization baseline. *Left:* $m_{\max} = 5$ (40 total solver macro-steps). *Right:* $m_{\max} = 8$ (52 total solver macro-steps). State vectorization consistently achieves the highest throughput by maximizing batch size per GPU kernel launch.

Analysis. State vectorization provides the dominant speedup, achieving $14.2\times$ at $m_{\max} = 5$ and $12.7\times$ at $m_{\max} = 8$ over the sequential baseline. Its advantage stems from maximizing the batch size per GPU kernel: at depth $j = 0$, all K branches are active, yielding a mega-batch of $K \times B = 40$ samples in a single `vmap` dispatch. As branches terminate at deeper levels, the batch size decreases gracefully.

Solver vectorization achieves a more modest 4.6–$4.7\times$ speedup. Although it eliminates inter-step bridge overhead by fusing each branch's chain into a single XLA program, each dispatch processes only $B = 4$ samples—an order of magnitude fewer than the state-vectorization mega-batch. The

$K = 10$ sequential dispatches (one per branch) also incur more total launch overhead than the $m_{\max}$ dispatches of state vectorization.

In practice, we adopt state vectorization as the default backend due to its consistently superior throughput and minimal implementation complexity. Its only requirement is that the solver supports batched evaluation, which is satisfied by any `jax.vmap`-compatible stepper.

D Model Architecture Details

We provide detailed descriptions of the six PDE foundation models evaluated in our experiments. Table 12 summarizes the architecture family, spatial support, core mechanism, pretraining strategy, fine-tuning setting, and parameter count for each model. All models are pretrained on PDE corpora and post-trained within our unified pipeline.

Table 12: Summary of the six PDE foundation models. **Params** reports the total parameter count of the instantiated wrapper used in our experiments. For DPOT-Ti, we list both the 2D and 3D parameter counts; the difference arises from the 3D patch embedding, positional embedding, and output reconstruction layers. All 2D models operate at 160^2 resolution; all 3D models at 64^3.

Model	Spatial	Architecture Core	Pretraining	Tuning	Params
PDE-S [1]	2D	Swin U-Net Transformer	16 PDEs	Full	33.2M
MPP-B [3]	2D	Axial ViT (AViT)	Multi-physics	Full	115.6M
DPOT-Ti [4]	2D / 3D	AFNO spectral blocks	AR denoising	Full	5.2M / 12.7M
Poseidon-T [2]	2D	SwinV2 U-Net (ScOT)	All-to-all semigroup	Full	20.7M
P3D-S [8]	3D	Conv3D + windowed 3D attn.	14 3D PDEs	Full	11.2M
MORPH-Ti [7]	3D	4D axial ViT + cross-attn.	Heterogeneous 1D–3D	Full	9.9M

PDE-S. PDE-Transformer [1] is a Swin-like U-Net Transformer. The architecture features patch-based input tokenization with patch size 4, a hierarchical encoder-decoder with depths $[2, 5] \to [8] \to [5, 2]$ and embed dimension 96, shifted-window attention with periodic padding support, and AdaLN-Zero conditioning on diffusion timestep and PDE class labels. The PDE-S variant uses 4 attention heads and operates at 160^2 resolution. We use the deterministic prediction mode (timestep forced to zero), and the PDE class index is set according to the target task.

MPP-B (Multiple Physics Pretraining). MPP [3] employs an Axial Vision Transformer (AViT) with interleaved spatial axial attention (along H and W axes independently) and temporal attention blocks operating along the time axis, each with relative positional biases. The model uses sparse per-physics input/output projections conditioned on state labels, integer indices into a pretrained vocabulary of physical quantities (e.g., velocity Vx/Vy). MPP-B has embed dimension 768, 12 attention heads, 12 processor blocks, and patch size $[16, 16]$. It was pretrained on 12 physical systems spanning fluid dynamics, electromagnetics, and solid mechanics.

DPOT-Ti (Denoising Operator Transformer). DPOT [4] is built on the Adaptive Fourier Neural Operator (AFNO) architecture. Each of its $d = 4$ spectral mixing blocks applies a learnable two-layer complex-valued MLP in the truncated Fourier domain, retaining only the lowest $k = 32$ frequency modes in 2D and $k = 8$ modes in our 3D configuration. This spectral filtering provides a global receptive field through FFT-based mixing. The architecture comprises: (1) a convolutional patch embed layer (patch size 8) that projects the input augmented with spatiotemporal coordinate channels into a latent space of dimension 512; (2) a time aggregator with learnable exponential-cosine temporal encoding; (3) a residual AFNO block stack with 1×1 convolutional feed-forward networks; and (4) a transposed-convolutional output layer that reconstructs the prediction at the original spatial resolution.

DPOT is the only model applied to both 2D and 3D tasks. The 2D variant (`DPOTNet`, 160^2) uses Conv2d operations with patch kernel size $8^2 = 64$ and contains 5.2M parameters. The 3D variant (`DPOTNet3D`, 64^3) keeps the same DPOT-Ti width, depth, and AFNO block count, but replaces the patch embedder, positional embedding, and output reconstruction with 3D counterparts. This increases the instantiated parameter count to 12.7M because the learned patch/output kernels scale from 8^2 to 8^3. Crucially, the AFNO spectral parameters (w_1, w_2, b_1, b_2) are shape-invariant between

2D and 3D and are directly transferred from the pretrained 2D checkpoint; the 1×1 block MLP weights are unsqueezed from Conv2d to Conv3d, while the 3D patch embedding, positional embedding, and output layer are randomly initialized and fine-tuned.

Poseidon-T. Poseidon [2] is built on the ScOT (Scalable Operator Transformer) architecture, adapting SwinV2 for PDE operator learning. It features 4 encoder-decoder stages with depths $[4, 4, 4, 4]$, embed dimension 48, attention heads $[3, 6, 12, 24]$, window size 16, and patch size 4. Poseidon was pretrained with an all-to-all semigroup strategy that forms supervised pairs across different lead times within each trajectory, not with autoregressive pushforward unrolling. In our wrapper, time conditioning is disabled (`use_conditioning=false`, `time=None`) to match the single-step pipeline. Poseidon includes built-in spectral (FFT-based) resolution adaptation: if the input resolution (160^2) differs from the pretrained image size (128^2), the model internally resamples via spectral truncation/padding. Since the released checkpoint uses 4 input/output channels whereas our 2D tasks use 2 effective channels, the embedding and recovery layers are reinitialized and trained together with the full model weights.

P3D-S (Physics-3D). P3D [8] is a hybrid CNN-Transformer architecture for scalable 3D PDE surrogate modelling. The model combines: (1) a conditioned 3D encoder with stride-2 Conv3D downsampling, AdaLN conditioning, and skip connections; (2) a `P3DBackbone` with windowed 3D attention using SwinV2-style relative positional biases, window size 4, and encoder/decoder stages connected by PixelShuffle3D up/downsampling; and (3) a conditioned 3D decoder that upsamples back to original resolution. P3D-S has a hidden size of 64, 5 stages with depth $[2, 2, 2, 2, 2]$, 4 heads per stage, and 2 convolutional downsampling layers. The pretrained model operates deterministically (timestep and class conditioning zeroed). P3D requires a minimum input resolution of 64^3 as its latent must be $\geq$ window size; for smaller inputs, the wrapper applies circular padding.

MORPH-Ti. MORPH [7] is a modality-agnostic PDE foundation model with a convolutional Vision Transformer architecture. It processes volumetric fields through: (1) a 3D hybrid patch embedding with Conv3D and inter-field cross-attention (32 heads); (2) spatio-temporal positional encoding; (3) $d = 4$ encoder blocks with *4D axial attention* decomposed along the temporal, depth, height, and width axes through four axial attention modules; and (4) a decoder that maps each patch token back to physical space via a linear projection. MORPH-Ti has latent dimension 256, 4 attention heads per axial module, MLP dimension 1024, and patch size 8. To mitigate blocky artefacts from the per-patch independent decoder, we added a lightweight residual Conv3D refinement head (zero-initialised) and enforced a minimum of 8 patches per spatial axis via circular padding.

Pipeline adaptations. Three models with native multi-timestep temporal input (DPOT, MPP, MORPH) are operated in single-frame mode ($T = 1$) to match our pipeline's single-step prediction interface. For DPOT, the pretrained $T = 10$ time aggregator is replaced by a $T = 1$ aggregator and re-learned during post-training. For MPP and MORPH, the temporal attention modules remain loaded from the pretrained checkpoint but operate over a singleton temporal axis; in MORPH, inter-field cross-attention also becomes singleton because the wrapper uses $F = 1$. The remaining three models (PDE-Transformer, Poseidon, P3D) natively process single spatial frames and require no temporal modification, although Poseidon's lead-time conditioning is explicitly disabled in our setup. Channel mismatches between pretrained weights and target tasks are handled differently per model: DPOT and Poseidon use random reinitialisation of embedding/recovery layers; P3D slices pretrained weights along the channel dimension to preserve learned features; MPP avoids channel mismatches through its sparse projections conditioned on state labels. Resolution mismatches are handled via bicubic interpolation of positional embeddings (DPOT-2D), built-in spectral resampling (Poseidon), or circular padding to meet minimum resolution requirements (P3D, MORPH).

D.1 Configuration of Experiments

All post-training runs are launched from the same composed configuration system. The training protocol, grid-search axes, and run-selection rule are summarized in appendix E.5; the present section focuses on the model-specific architecture and wrapper details needed to instantiate each pretrained backbone.

E Experimental Details

We describe the governing equations, solver methodology, and dataset generation parameters for all seven PDE benchmark tasks. All PDEs are solved on periodic domains using the Exponax library [47], which implements Exponential Time Differencing Runge–Kutta (ETDRK) pseudo-spectral methods in JAX. Initial conditions are random truncated Fourier series with wavenumber cutoff $k_{\max} = 5$.

E.1 Solver Methodology

Exponax solves semi-linear PDEs of the form $\partial_t u = \hat{\mathcal{L}} u + \hat{\mathcal{N}}(u)$, where $\hat{\mathcal{L}}$ is a linear operator and handled exactly in Fourier space via the matrix exponential, while $\hat{\mathcal{N}}$ is a nonlinear term. Spatial derivatives are computed pseudo-spectrally using FFTs with $2/3$-rule dealiasing. For stiff PDEs requiring fine temporal resolution, each macro time step Δt_{FM} is subdivided into N_{sub} internal ETDRK2 sub-steps of size $\delta t = \Delta t_{\mathrm{FM}}/N_{\mathrm{sub}}$, with the macro step realized as: $\mathcal{S}_{\Delta t} = \underbrace{\mathcal{S}_{\delta t} \circ \cdots \circ \mathcal{S}_{\delta t}}_{N_{\mathrm{sub}}}$.

E.2 Details of system equations

The Kuramoto–Sivashinsky and Burgers equations are specified through APEBench's *difficulty parameterization* [47]. The underlying Exponax stepper solves

$$\partial_t u \;=\; \sum_{j=0}^{J} a_j \,(\mathbf{1}\cdot\nabla^j)\, u \;+\; b_0\, u^2 \;+\; b_1\, \tfrac{1}{2}(\mathbf{1}\cdot\nabla)(u^2) \;+\; b_2\, \tfrac{1}{2}\|\nabla u\|_2^2, \tag{E.1}$$

where $(\mathbf{1}\cdot\nabla^j)\, u = \sum_{i=1}^{d} \partial_{x_i}^j u$, so the linear operators contain no cross-derivatives. On an N^d grid of a periodic domain with side L and time step Δt, the difficulties are dimensionless versions of these coefficients:

$$\gamma_j = \frac{a_j\,\Delta t}{L^j}\, N^j\, 2^{j-1} D \;\; (j \geq 1), \qquad \delta_1 = \frac{b_1\,\Delta t}{L}\, MND, \qquad \delta_2 = \frac{b_2\,\Delta t}{L^2}\, MN^2 D, \tag{E.2}$$

with $\gamma_0 = a_0 \Delta t$, $\delta_0 = b_0 \Delta t$, D the number of spatial dimensions, and $M = 1$ the expected maximum absolute value of the state. The stepper runs on $(0,1)^d$ with a unit time step, so $\boldsymbol{\gamma}$ and $\boldsymbol{\delta} = (\delta_0, \delta_1, \delta_2)$ fully determine the dynamics relative to the grid. Because the difficulties include N and D, the same values describe different continuum problems at 160^2 and at 64^3.

The Navier–Stokes benchmarks use Exponax's physics-based stepper classes, which directly parameterize the governing equations through physical quantities (viscosity, Reynolds number, forcing parameters) rather than abstract difficulty coefficients.

Kuramoto–Sivashinsky (KS2D, KS3D). The KS equation models spatiotemporal chaos arising from the interplay of anti-diffusion (energy production at intermediate scales) and hyper-diffusion (energy dissipation at small scales):

$$\partial_t u + \tfrac{b}{2}\|\nabla u\|_2^2 = -\nu\,\Delta u - \zeta \sum_{i=1}^{d} \partial_{x_i}^4 u, \tag{E.3}$$

with $d \in \{2, 3\}$ and coefficients set by the difficulties through eq. (E.2). We use the flame-front (gradient-norm) form of the nonlinearity rather than the convective form $u \cdot \nabla u$; the two coincide in 1D under $u = \partial_x \phi$ but are distinct operators in higher dimensions, and Exponax's difficulty interface parameterizes the gradient-norm term. In the difficulty parameterization, this corresponds to $\boldsymbol{\gamma} = (0, 0, -1.2, 0, -15)$ and $\boldsymbol{\delta} = (0, 0, -6)$, where $\gamma_2 = -1.2$ provides anti-diffusion, $\gamma_4 = -15$ provides hyper-diffusion, and $\delta_2 = -6$ scales the gradient-norm nonlinearity. Up to a rescaling of u, these values correspond to the classical KS equation ($b = \nu = \zeta = 1$) on a periodic box of side $\ell \approx 0.566\, N$, i.e., $\ell \approx 90.5$ for KS2D ($N = 160$) and $\ell \approx 36.2$ for KS3D ($N = 64$); on $[0, 2\pi]^d$ the classical equation has no linearly unstable mode.

Burgers (Burgers2D, Burgers3D). The single-channel Burgers equation combines viscous diffusion with a convective nonlinearity acting along the diagonal direction $\mathbf{1}$:

$$\partial_t u + \tfrac{1}{2}(\mathbf{1}\cdot\nabla)(u^2) = \nu\,\Delta u, \tag{E.4}$$

where $\frac{1}{2}(\mathbf{1}\cdot\nabla)(u^2) = u\sum_{i=1}^{d}\partial_{x_i}u$. The difficulty parameterization is $\boldsymbol{\gamma} = (0, 0, 1.5, 0, 0)$ and $\boldsymbol{\delta} = (0, -1.5, 0)$, where $\gamma_2 = 1.5$ provides forward diffusion and $\delta_1 = -1.5$ encodes the convective nonlinearity. By eq. (E.2), the Reynolds number based on the domain size and the maximum initial amplitude is $2N|\delta_1|/\gamma_2 = 2N$, i.e., 320 for Burgers2D and 128 for Burgers3D.

Kolmogorov Flow — Vorticity Formulation (KF2D). Kolmogorov flow solves the 2D incompressible Navier–Stokes equations in vorticity formulation with a monochromatic body force. The vorticity $\omega = \nabla \times \mathbf{v}$ evolves as:

$$\partial_t\omega + (\mathbf{v}\cdot\nabla)\omega = \nu\nabla^2\omega + \mu\omega + f, \qquad f = -\tfrac{2\pi n_f}{L}\,A\cos\Big(\tfrac{2\pi n_f}{L}y\Big), \tag{E.5}$$

where ν is the kinematic viscosity, $\mu < 0$ is a linear drag coefficient that removes energy at the largest scales, A is the forcing amplitude, and n_f is the forcing wavenumber (injection mode). The forcing f is the curl of the Kolmogorov body force $A\sin(2\pi n_f y/L)\,\mathbf{e}_x$ acting on the primary variables, which is why it enters the vorticity equation as a cosine carrying the extra factor $2\pi n_f/L$. The velocity field $\mathbf{v}$ is recovered from the vorticity via the Biot–Savart law in Fourier space. The KF2D task uses $\nu = 0.01$ ($\mathrm{Re} \approx 100$), $\mu = -0.2$, forcing mode $n_f = 4$, forcing amplitude $A = 1.0$, and $L = 2\pi$, so $f = -4\cos(4y)$. The APEBench data were generated with Exponax 0.1.0 and drag parameter -0.1, which that version applies with a factor of D, giving $\mu = -0.2$ in 2D.

Kolmogorov Flow — Velocity Formulation (KF3D). The 3D Kolmogorov flow is solved directly in velocity formulation, evolving the incompressible velocity field $\mathbf{v} = (v_x, v_y, v_z)$:

$$\partial_t\mathbf{v} + (\mathbf{v}\cdot\nabla)\mathbf{v} = -\nabla p + \nu\nabla^2\mathbf{v} + \mu\,\mathbf{v} + \mathbf{f}, \qquad \nabla\cdot\mathbf{v} = 0, \tag{E.6}$$

where p is the pressure enforcing incompressibility (projected out in Fourier space via the Leray projector), and $\mathbf{f} = A\sin(2\pi n_f y/L)\,\mathbf{e}_x$ is a sinusoidal body force. The KF3D task uses $\nu = 0.01$, $\mu = -0.1$, $n_f = 4$, $A = 1$, and $L = 2\pi$. The velocity formulation produces 3 output channels (v_x, v_y, v_z), and initial conditions are projected to be divergence-free via `ex.spectral.make_incompressible`. A 200-step warm-up period drives the flow into a statistically stationary turbulent regime before data recording begins.

Decaying Turbulence — Velocity Formulation (DT3D). Decaying turbulence solves the unforced 3D incompressible Navier–Stokes equations:

$$\partial_t\mathbf{v} + (\mathbf{v}\cdot\nabla)\mathbf{v} = -\nabla p + \nu\nabla^2\mathbf{v}, \qquad \nabla\cdot\mathbf{v} = 0, \tag{E.7}$$

without body force or linear drag. Energy injected through the initial condition cascades from large to small scales and is dissipated by viscosity, resulting in a monotonically decaying energy spectrum. This task tests the model's ability to capture the temporal evolution of spectral energy transfer without the stationarity provided by external forcing.

E.3 Dataset Generation and Solver Parameters

All 2D datasets are sourced from the original APEBench benchmark [47], while the 3D datasets are generated at higher resolution (64^3) by directly instantiating the corresponding Exponax stepper. Table 13 summarizes the complete solver parameters for all seven benchmark tasks.

Training data and splits. Each in-distribution task uses 50 training trajectories of 50 time steps and a held-out test set of 200-step trajectories, all generated from independent random initial conditions drawn from a truncated Fourier series with wavenumber cutoff $k_{\max} = 5$. All 50 training trajectories are used for training. During post-training we monitor the rollout RMSE on the held-out trajectories at horizons 10 and 30: the plateau scheduler uses the 30-step value, and the checkpoint with the lowest 10-step value is retained. All methods use the same protocol. Reported metrics, in distribution and OOD, are 50-step autoregressive rollouts from the first five test trajectories.

Warm-up protocol. For the Kolmogorov flow tasks, the sinusoidal body forcing must first drive the system into a statistically stationary turbulent regime; we therefore discard a warm-up transient before recording data: 500 steps for KF2D and 200 steps for KF3D. Similarly, for Kuramoto–Sivashinsky (KS2D, KS3D), a 500-step warm-up develops the characteristic spatiotemporal chaos, as

Table 13: Solver parameters for all seven PDE benchmark tasks. **Top**: 2D tasks at 160^2 resolution. **Bottom**: 3D tasks at 64^3 resolution. Δt_{FM} is the macro time step per FM prediction, N_{sub} is the number of ETDRK2 sub-steps per macro step, and C is the number of data channels.

Task	PDE	C	Resolution	Δt_{FM}	N_{sub}	Key Parameters
KS2D	Kuramoto–Sivashinsky	1	160^2	1.0	1	$\boldsymbol{\gamma}=(0,0,-1.2,0,-15)$; $\boldsymbol{\delta}=(0,0,-6)$
Burgers2D	Burgers (single-ch.)	1	160^2	1.0	1	$\boldsymbol{\gamma}=(0,0,1.5,0,0)$; $\boldsymbol{\delta}=(0,-1.5,0)$
KF2D	Kolmogorov Flow (vort.)	1	160^2	0.1	20	$\nu=0.01$, drag$=-0.1$, mode$=4$, $L=2\pi$
KS3D	Kuramoto–Sivashinsky	1	64^3	1.0	1	$\boldsymbol{\gamma}=(0,0,-1.2,0,-15)$; $\boldsymbol{\delta}=(0,0,-6)$
Burgers3D	Burgers (single-ch.)	1	64^3	1.0	1	$\boldsymbol{\gamma}=(0,0,1.5,0,0)$; $\boldsymbol{\delta}=(0,-1.5,0)$
KF3D	Kolmogorov Flow (vel.)	3	64^3	0.1	20	$\nu=0.01$, drag$=-0.1$, mode$=4$, $L=2\pi$
DT3D	Decaying Turb. (vel.)	3	64^3	0.1	20	$\nu=0.01$, drag$=0$, $L=2\pi$

the energy-producing anti-diffusion requires time to populate the chaotic attractor from the smooth random Fourier initial conditions. Both Burgers tasks and decaying turbulence (DT3D) use no warm-up: Burgers' positive diffusion immediately steepens the initial condition into shock fronts, so the transient formation is itself the target dynamics; decaying turbulence captures the energy cascade from the initial vortex field, which is likewise the phenomenon of interest. These conventions follow the APEBench defaults; the KF3D warm-up is our own choice, since APEBench has no 3D Kolmogorov scenario.

E.4 Out-of-Distribution Zero-Shot Datasets

To assess generalization beyond the training distribution, we evaluate post-trained models on out-of-distribution (OOD) test sets generated by perturbing the physical parameters of the in-distribution tasks. These datasets are test-only (10 trajectories, 200 steps each) and share the same spatial resolution and domain as their in-distribution counterparts. The OOD configurations are generated via the same Exponax solver infrastructure. The lower viscosity variant ($\nu = 0.005$) requires doubled sub-stepping ($N_{\mathrm{sub}} = 40$) to maintain solver stability. Table 14 summarizes the parameter perturbations for each OOD task.

Table 14: Out-of-distribution (OOD) zero-shot evaluation datasets. Each row lists the parameter changes relative to the in-distribution training task.

OOD Task	Base Task	Parameter Changes
2D tasks (160^2)		
Burgers2D-OOD	Burgers2D	$\gamma_2: 1.5 \to 0.5$; $\delta_1: -1.5 \to -0.5$
KF2D-OOD	KF2D	$\nu: 0.01 \to 0.005$ ($N_{\mathrm{sub}}: 20 \to 40$); linear drag halved
KS2D-OOD	KS2D	$\delta_2: -6 \to -8$
3D tasks (64^3)		
Burgers3D-OOD	Burgers3D	$\delta_1: -1.5 \to -0.5$
KF3D-OOD	KF3D	$\nu: 0.01 \to 0.005$ ($N_{\mathrm{sub}}: 20 \to 40$)
KS3D-OOD	KS3D	$\delta_2: -6 \to -8$
DT3D-OOD	DT3D	$\nu: 0.01 \to 0.02$

E.5 Training strategy

To ensure a rigorous empirical comparison, we employ a systematic hyperparameter sweep rather than relying on a single, hand-tuned configuration. For each available model–task–method pair, we

perform a grid search over a defined hyperparameter space. For each combination we then report the best-performing run of this search. This strict isolation guarantees that each method is evaluated at its peak capacity for a specific model and task, preventing the bias introduced by transferring suboptimal hyperparameters across disparate architectures.

Every run follows the same two-stage structure on an NVIDIA A100 GPU. First, an embedded SFT warm-up adapts the pretrained backbone to the downstream PDE distribution using single-step supervision ($K = 1$), with 80 warm-up epochs for 2D runs and 40 warm-up epochs for 3D runs. Second, the main phase applies the method-specific objective: SFT trains against reference trajectories, PITA trains with its physics-informed residual objective, OPD uses the same on-policy rollout loop as SaT but replaces the numerical solver with a larger FM teacher, and SaT trains with the unified solver/anchor objective in eq. (3). All methods share the same data split, trainer, checkpointing logic, rollout evaluation code, AdamW optimizer, gradient clipping at $\|\nabla\|_{\max} = 1.0$, and composite per-step loss with MSE weight 1.0 and Sobolev H^1 weight 0.5 when such supervised targets are used.

Table 15: Training grid used for model–task–method selection. The table lists the search axes used by the grid scripts and the realized ranges recorded in the complete run summary; individual retained configurations can be recovered from `experiment_details_2d.md`.

Axis	Values / protocol
Methods	SFT, PITA (2D only), OPD, SaT-1 ($\alpha = 1$), SaT-.5 ($\alpha = 0.5$), SaT-0 ($\alpha = 0$)
Optimization	AdamW; full fine-tuning in the standard grid; bf16 mixed precision
Learning rate	10^{-3}
Epoch	160 for 3D and 320 for 2D
Scheduler	Plateau or cosine scheduling, with 5% warm-up and plateau factor 0.5 when plateau is used
Batching	Base batch size is 6 for 3D and 12 for 2D; combined with gradient accumulation in $\{1, 2, 4, 8\}$
On-policy branching	SaT/OPD use $m_{\max} = 5$ multi-step lookahead; λ and γ are set to 0.9

OPD requires an additional teacher-preparation step. Since $K_{\max} = 5$, we first adapt the larger full-size teacher backbone with the same two-stage strategy used for the students: a single-step SFT warm-up, followed by an unrolled SFT main phase whose rollout horizon increases linearly from $K = 1$ to $K_{\max} = 5$. This gives OPD a task-specialized teacher that has already been trained for the same autoregressive horizon used in the student grid, avoiding a comparison against an unadapted large pretrained model. During the subsequent OPD run, this teacher checkpoint is frozen and queried on the student-visited states; only the student parameters are updated. For SaT, the same rollout horizon and batching axes are used, but teacher targets are generated by Exponax from the student-visited states with the vectorized solver backend described in appendix C. After post-training, each retained checkpoint is evaluated with a 50-step autoregressive rollout on the first five held-out test trajectories using the common inference engine.

F Additional Physical Diagnostics

The main evaluation uses rollout RMSE as the primary accuracy metric, since it directly measures pointwise deviation from the held-out solver trajectory. RMSE alone, however, does not distinguish a uniformly small amplitude error from a loss of spatial gradients or a distorted physical property like spectral energy distribution. We therefore report two additional trajectory-level diagnostics on the same autoregressive rollouts: a Sobolev H^1 error, which emphasizes gradient-scale mismatch, and a normalized energy-spectrum error.

All three metrics compare a predicted rollout $\hat{\tau} = \{\hat{u}_1, \dots, \hat{u}_K\}$ against the corresponding reference $\tau^{\text{ref}} = \{u_1^{\text{ref}}, \dots, u_K^{\text{ref}}\}$. Metrics are averaged over all K rollout steps and B test samples; for notational brevity, we write $N = BK$ and let a single index $i \in \{1, \dots, N\}$ range over the flattened sample–time axis.

The trajectory-level RMSE is

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left\|\hat{u}_i - u_i^{\mathrm{ref}}\right\|^2}, \tag{F.1}$$

where $\|\cdot\|$ denotes the ℓ_2 norm over all channels and grid points, normalised by the total number of elements (i.e. $\|v\|^2 = \frac{1}{C|\Omega_h|}\sum_{c,\mathbf{x}} |v_c(\mathbf{x})|^2$).

For the Sobolev diagnostic, let $e_i = \hat{u}_i - u_i^{\mathrm{ref}}$ denote the error field and $\widehat{e}_i(\boldsymbol{\nu})$ its orthonormalized discrete Fourier transform at normalized frequency $\boldsymbol{\nu}$ ($\nu_j = q_j/N_j$, with integer wavenumber q_j and grid size N_j along axis j). The first-order Sobolev error is

$$\mathcal{E}_{H^1} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\frac{1}{C|\Omega_h|}\sum_{c}\sum_{\boldsymbol{\nu}}\left(1+|\boldsymbol{\nu}|^2\right)\left|\widehat{e}_{i,c}(\boldsymbol{\nu})\right|^2}. \tag{F.2}$$

By Parseval's theorem with orthonormalized Fourier coefficients, $\sum_{\boldsymbol{\nu}} |\widehat{e}_{i,c}|^2 = \sum_{\mathbf{x}} |e_{i,c}|^2$, so setting $s = 0$ recovers RMSE exactly. At $s = 1$ the factor $(1 + |\boldsymbol{\nu}|^2)$ indicates unresolved fine-scale structure more strongly than RMSE.

The energy-spectrum diagnostic compares the radial distribution of Fourier energy. For a field z_i, let $\widehat{z}_{i,c}(\boldsymbol{\nu})$ denote the un-normalized DFT coefficient of channel c. Grouping modes into radial bins $\mathcal{B}_r$ by their normalized-frequency magnitude, the per-sample spectrum and its normalization are

$$E_z^i(r) = \frac{1}{2}\sum_{c}\sum_{\boldsymbol{\nu}\in\mathcal{B}_r}\left|\widehat{z}_{i,c}(\boldsymbol{\nu})\right|^2, \qquad \widetilde{E}_z^i(r) = \frac{E_z^i(r)}{\sum_{r'} E_z^i(r')}. \tag{F.3}$$

Each of the N spectra is normalized to sum to one before averaging: $\bar{E}_z(r) = N^{-1}\sum_i \widetilde{E}_z^i(r)$. The reported metric is the relative discrepancy

$$\mathcal{E}_{\mathrm{spec}} = \frac{\left\|\bar{E}_{\hat{u}} - \bar{E}_{u^{\mathrm{ref}}}\right\|_2}{\left\|\bar{E}_{u^{\mathrm{ref}}}\right\|_2}. \tag{F.4}$$

Because the spectra are normalized per sample before comparison, $\mathcal{E}_{\mathrm{spec}}$ focuses on spectral *shape* rather than absolute amplitude. It is therefore complementary to both RMSE and $\mathcal{E}_{H^1}$: RMSE measures pointwise tracking, H^1 measures fine-scale structure error, and the spectrum error probes whether the rollout preserves the scale-wise energy distribution expected from the governing dynamics.

Table 16 reports the same diagnostics under parameter shifts. SaT-0 is best on all seven Sobolev H^1 rows, which is consistent with the main text's conclusion that full dynamic loss is the most robust choice once the model is evaluated away from the training distribution. Spectrum errors are more sensitive to the specific PDE and parameter shift, but a SaT variant is still best on six of seven OOD tasks; the only exception is Burgers2D-OOD, where OPD has the lowest spectrum error. This indicates that solver-guided on-policy training improves not only pointwise tracking but also the scale-wise statistics needed for credible chaotic and turbulent rollouts.

G Theoretical Analysis

We formalize the distribution-alignment advantage of Solver-as-Teacher (SaT). In particular, we show that the dynamics component of the SaT objective directly controls the one-step solver-correction error on student-visited states. Combined with a standard no-regret online-learning argument and a stability assumption on the numerical solver, this yields a bound on the autoregressive rollout error.

The analysis is stated for the DAgger-style interpretation of SaT. At iteration n, the current model G_n first generates a rollout from an initial condition $u_0 \sim \rho_0$:

$$\hat{u}_{n,0} = u_0, \qquad \hat{u}_{n,k+1} = G_n(\hat{u}_{n,k}), \qquad k = 0, \ldots, K-1. \tag{G.1}$$

Let $d_{G_n}^k$ denote the distribution of $\hat{u}_{n,k}$. The corresponding reference trajectory is generated by the solver:

$$u_0^{\mathrm{ref}} = u_0, \qquad u_{k+1}^{\mathrm{ref}} = S(u_k^{\mathrm{ref}}). \tag{G.2}$$

Table 16: Cross-model aggregate physical diagnostics on OOD zero-shot test sets. Metrics are mean ± std over evaluated architectures; lower is better. Best per row is **bolded**. PITA does not support 3D tasks (–).

Task	Metric	SFT	PITA	OPD	SaT-1	SaT-.5	SaT-0	N models
Burgers2D-OOD	H^1	0.0857 ± 0.0157	0.0771 ± 0.0079	0.0739 ± 0.0016	0.0727 ± 0.0014	0.0729 ± 0.0021	$\mathbf{0.0723 \pm 0.0010}$	4
	Spectrum	0.1939 ± 0.0202	0.1860 ± 0.0214	$\mathbf{0.1852 \pm 0.0117}$	0.1918 ± 0.0072	0.1884 ± 0.0117	0.1893 ± 0.0077	
KF2D-OOD	H^1	2.4544 ± 0.2901	2.4703 ± 0.2257	2.4244 ± 0.2118	2.3333 ± 0.1739	2.5008 ± 0.2599	$\mathbf{2.2669 \pm 0.2140}$	4
	Spectrum	0.1959 ± 0.0296	0.2027 ± 0.0272	0.2226 ± 0.0133	0.2134 ± 0.0216	$\mathbf{0.1911 \pm 0.0176}$	0.2147 ± 0.0197	
KS2D-OOD	H^1	0.1376 ± 0.0311	0.1203 ± 0.0348	0.1265 ± 0.0282	0.1253 ± 0.0307	0.1072 ± 0.0262	$\mathbf{0.1063 \pm 0.0176}$	4
	Spectrum	0.2416 ± 0.1864	0.1222 ± 0.0465	0.2181 ± 0.2132	0.2538 ± 0.2230	$\mathbf{0.0967 \pm 0.0510}$	0.1365 ± 0.0968	
Burgers3D-OOD	H^1	0.0177 ± 0.0026	–	0.0144 ± 0.0020	0.0132 ± 0.0012	0.0142 ± 0.0020	$\mathbf{0.0130 \pm 0.0010}$	3
	Spectrum	0.0803 ± 0.0394	–	0.0380 ± 0.0039	0.0418 ± 0.0234	0.0298 ± 0.0030	$\mathbf{0.0297 \pm 0.0118}$	
KF3D-OOD	H^1	1.0929 ± 1.2117	–	0.2036 ± 0.0055	0.2004 ± 0.0042	0.1980 ± 0.0049	$\mathbf{0.1945 \pm 0.0045}$	3
	Spectrum	0.2196 ± 0.1327	–	0.1562 ± 0.0498	$\mathbf{0.1317 \pm 0.0258}$	0.1335 ± 0.0161	0.1319 ± 0.0230	
KS3D-OOD	H^1	2.6867 ± 3.5221	–	0.2910 ± 0.0965	0.2016 ± 0.0189	0.1966 ± 0.0169	$\mathbf{0.1929 \pm 0.0228}$	3
	Spectrum	0.1683 ± 0.1695	–	0.0944 ± 0.0161	0.0876 ± 0.0569	0.0723 ± 0.0445	$\mathbf{0.0690 \pm 0.0507}$	
DT3D-OOD	H^1	0.2520 ± 0.2880	–	0.0438 ± 0.0115	0.0428 ± 0.0064	0.0436 ± 0.0077	$\mathbf{0.0372 \pm 0.0028}$	3
	Spectrum	0.3981 ± 0.0732	–	0.3301 ± 0.0335	0.3178 ± 0.0207	$\mathbf{0.3024 \pm 0.0174}$	0.3060 ± 0.0158	

For each student-visited state $\hat{u}_{n,k}$, define

$$m_k := \min(m_{\max}, K - k). \tag{G.3}$$

After collecting the rollout, the states are treated as fixed training inputs. For any candidate model $G \in \mathcal{G}$, let G^j denote the j-fold composition of G, and define the round-n dynamics and anchor losses by

$$\mathcal{L}_{\mathrm{dyn},n}(G) := \frac{1}{W_{\mathrm{dyn}}} \sum_{k=0}^{K-1} \sum_{j=1}^{m_k} \gamma^k \lambda^{j-1} \mathbb{E}_{u_0 \sim \rho_0} \left[\ell \left(G^j(\hat{u}_{n,k}), S^j(\hat{u}_{n,k}) \right) \right], \tag{G.4}$$

$$\mathcal{L}_{\mathrm{anchor},n}(G) := \frac{1}{W_{\mathrm{anchor}}} \sum_{k=0}^{K-1} \gamma^k \mathbb{E}_{u_0 \sim \rho_0} \left[\ell \left(G(\hat{u}_{n,k}), u_{k+1}^{\mathrm{ref}} \right) \right]. \tag{G.5}$$

Here W_{dyn} and W_{anchor} are the same normalization constants as in the main SaT objective. The complete round-n loss is

$$\mathcal{L}_n^{\alpha}(G) := (1 - \alpha)\mathcal{L}_{\mathrm{dyn},n}(G) + \alpha \mathcal{L}_{\mathrm{anchor},n}(G), \qquad \alpha \in [0, 1]. \tag{G.6}$$

When evaluated at $G = G_n$, this is precisely the expected version of the training objective in the main text, since

$$G_n^j(\hat{u}_{n,k}) = \hat{u}_{n,k+j}. \tag{G.7}$$

We define the one-step on-policy solver-correction error of the n-th iterate as

$$\mathcal{E}_{\mathrm{on}}(G_n) := \frac{1}{K} \sum_{k=0}^{K-1} \mathbb{E}_{\hat{u}_{n,k} \sim d_{G_n}^k} \left[\ell \left(G_n(\hat{u}_{n,k}), S(\hat{u}_{n,k}) \right) \right]. \tag{G.8}$$

For the sequence of online losses $\{\mathcal{L}_n^{\alpha}\}_{n=1}^N$, define the best-in-hindsight approximation error

$$\varepsilon_N^{\alpha} := \min_{G \in \mathcal{G}} \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}_n^{\alpha}(G), \tag{G.9}$$

and the online learning regret

$$R_N := \sum_{n=1}^{N} \mathcal{L}_n^{\alpha}(G_n) - \min_{G \in \mathcal{G}} \sum_{n=1}^{N} \mathcal{L}_n^{\alpha}(G). \tag{G.10}$$

Finally, for a fixed model G, let $\hat{u}_{k+1} = G(\hat{u}_k)$ and $u_{k+1}^{\mathrm{ref}} = S(u_k^{\mathrm{ref}})$, with $\hat{u}_0 = u_0^{\mathrm{ref}}$. We define the rollout error by

$$\mathcal{E}_{\mathrm{SaT}}(G; S) := \left(\frac{1}{K} \sum_{k=1}^{K} \mathbb{E} \left[\left\| \hat{u}_k - u_k^{\mathrm{ref}} \right\|_{L^2(\Omega)}^2 \right] \right)^{1/2}. \tag{G.11}$$

This is the population analogue of the trajectory-level rollout RMSE used in the experiments.

Theorem G.1 (On-policy and rollout guarantees for SaT)**.** *Suppose that $m_{\max} \geq 1$, $\gamma, \lambda \in (0, 1]$, and $\alpha \in [0, 1)$. Assume that the solver transition is L_S-Lipschitz on the states visited by the solver and student rollouts:*

$$\|S(u) - S(v)\|_{L^2(\Omega)} \leq L_S \|u - v\|_{L^2(\Omega)}. \tag{G.12}$$

The composite loss in the main text satisfies

$$\ell(v, w) \geq \beta_1 \|v - w\|^2_{L^2(\Omega)}, \tag{G.13}$$

because its Sobolev component is nonnegative.

Define

$$\kappa_{K,m_{\max},\gamma,\lambda} := \frac{W_{\mathrm{dyn}}}{K\gamma^{K-1}}. \tag{G.14}$$

Then there exists an iterate $\hat{n} \in \{1, \ldots, N\}$ such that

$$\mathcal{E}_{\mathrm{on}}(G_{\hat{n}}) \leq \frac{\kappa_{K,m_{\max},\gamma,\lambda}}{1-\alpha} \left(\varepsilon_N^{\alpha} + \frac{R_N}{N} \right). \tag{G.15}$$

Moreover, the same iterate satisfies

$$\mathcal{E}_{\mathrm{SaT}}(G_{\hat{n}}; S) \leq \left(\sum_{r=0}^{K-1} L_S^r \right) \sqrt{\frac{\kappa_{K,m_{\max},\gamma,\lambda}}{\beta_1(1-\alpha)} \left(\varepsilon_N^{\alpha} + \frac{R_N}{N} \right)}. \tag{G.16}$$

Proof. We first show that the SaT training objective controls the one-step on-policy correction error. Since all loss terms are nonnegative and $m_k \geq 1$, the $j = 1$ terms of the dynamics loss give

$$\begin{aligned} \mathcal{L}_n^{\alpha}(G_n) &= (1-\alpha)\mathcal{L}_{\mathrm{dyn},n}(G_n) + \alpha\mathcal{L}_{\mathrm{anchor},n}(G_n) \\ &\geq \frac{1-\alpha}{W_{\mathrm{dyn}}} \sum_{k=0}^{K-1} \gamma^k \mathbb{E}\left[\ell\left(G_n(\hat{u}_{n,k}), S(\hat{u}_{n,k}) \right) \right]. \end{aligned} \tag{G.17}$$

Because $\gamma \in (0, 1]$, we have $\gamma^k \geq \gamma^{K-1}$ for every $k \in \{0, \ldots, K-1\}$. Hence

$$\begin{aligned} \mathcal{L}_n^{\alpha}(G_n) &\geq \frac{(1-\alpha)\gamma^{K-1}}{W_{\mathrm{dyn}}} \sum_{k=0}^{K-1} \mathbb{E}\left[\ell\left(G_n(\hat{u}_{n,k}), S(\hat{u}_{n,k}) \right) \right] \\ &= \frac{(1-\alpha)K\gamma^{K-1}}{W_{\mathrm{dyn}}} \mathcal{E}_{\mathrm{on}}(G_n). \end{aligned} \tag{G.18}$$

Rearranging and using (G.14), we obtain

$$\mathcal{E}_{\mathrm{on}}(G_n) \leq \frac{\kappa_{K,m_{\max},\gamma,\lambda}}{1-\alpha} \mathcal{L}_n^{\alpha}(G_n). \tag{G.19}$$

Averaging (G.19) over $n = 1, \ldots, N$ gives

$$\begin{aligned} \frac{1}{N} \sum_{n=1}^{N} \mathcal{E}_{\mathrm{on}}(G_n) &\leq \frac{\kappa_{K,m_{\max},\gamma,\lambda}}{1-\alpha} \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}_n^{\alpha}(G_n) \\ &= \frac{\kappa_{K,m_{\max},\gamma,\lambda}}{1-\alpha} \left[\min_{G \in \mathcal{G}} \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}_n^{\alpha}(G) + \frac{R_N}{N} \right] \\ &= \frac{\kappa_{K,m_{\max},\gamma,\lambda}}{1-\alpha} \left(\varepsilon_N^{\alpha} + \frac{R_N}{N} \right). \end{aligned} \tag{G.20}$$

Since the minimum of N nonnegative numbers is no larger than their average, there exists $\hat{n} \in \{1, \ldots, N\}$ satisfying (G.15).

It remains to relate the one-step correction error to rollout RMSE. Fix $G = G_{\hat{n}}$, and define

$$e_k := \left\| \hat{u}_k - u_k^{\mathrm{ref}} \right\|_{L^2(\Omega)}, \qquad \delta_k := \|G(\hat{u}_k) - S(\hat{u}_k)\|_{L^2(\Omega)}. \tag{G.21}$$

Using the student and solver recursions together with (G.12), we have

$$\begin{aligned} e_{k+1} &= \left\| G(\hat{u}_k) - S(u_k^{\mathrm{ref}}) \right\|_{L^2(\Omega)} \\ &\le \|G(\hat{u}_k) - S(\hat{u}_k)\|_{L^2(\Omega)} + \left\| S(\hat{u}_k) - S(u_k^{\mathrm{ref}}) \right\|_{L^2(\Omega)} \\ &\le \delta_k + L_S e_k. \end{aligned} \tag{G.22}$$

Since $e_0 = 0$, recursively applying (G.22) yields

$$e_k \le \sum_{j=0}^{k-1} L_S^{k-1-j} \delta_j. \tag{G.23}$$

The right-hand side is a causal discrete convolution between $\{\delta_j\}$ and the kernel $\{1, L_S, \dots, L_S^{K-1}\}$. By the discrete Young inequality,

$$\left(\sum_{k=1}^{K} e_k^2 \right)^{1/2} \le \left(\sum_{r=0}^{K-1} L_S^r \right) \left(\sum_{j=0}^{K-1} \delta_j^2 \right)^{1/2}. \tag{G.24}$$

Squaring, taking expectations, and dividing by K gives

$$\mathcal{E}_{\mathrm{SaT}}^2(G; S) \le \left(\sum_{r=0}^{K-1} L_S^r \right)^2 \frac{1}{K} \sum_{j=0}^{K-1} \mathbb{E}[\delta_j^2]. \tag{G.25}$$

By (G.13),

$$\begin{aligned} \frac{1}{K} \sum_{j=0}^{K-1} \mathbb{E}[\delta_j^2] &\le \frac{1}{\beta_1 K} \sum_{j=0}^{K-1} \mathbb{E}\left[\ell\left(G(\hat{u}_j), S(\hat{u}_j) \right) \right] \\ &= \frac{1}{\beta_1} \mathcal{E}_{\mathrm{on}}(G). \end{aligned} \tag{G.26}$$

Combining (G.25) and (G.26), and then taking square roots, gives

$$\mathcal{E}_{\mathrm{SaT}}(G; S) \le \left(\sum_{r=0}^{K-1} L_S^r \right) \sqrt{\frac{\mathcal{E}_{\mathrm{on}}(G)}{\beta_1}}. \tag{G.27}$$

Substituting (G.15) into (G.27) proves (G.16).

Remark G.1 (Dependence on the anchor coefficient)**.** The restriction $\alpha < 1$ is essential. The guarantee is obtained from the dynamics component of the objective, whose coefficient is $1 - \alpha$. When $\alpha = 1$, this component vanishes, and the objective reduces to SaT-1, namely uniformly weighted unrolled SFT with no on-policy solver-correction signal. Accordingly, the bound becomes vacuous at $\alpha = 1$. In its explicit dependence on α, the bound is strongest for SaT-0 and becomes weaker as more weight is assigned to the reference anchor.

Remark G.2 (Explicit form of the transfer constant)**.** If the dynamics normalizer is chosen as the sum of the dynamic weights,

$$W_{\mathrm{dyn}} = \sum_{k=0}^{K-1} \sum_{j=1}^{m_k} \gamma^k \lambda^{j-1}, \tag{G.28}$$

then the transfer constant becomes

$$\kappa_{K, m_{\max}, \gamma, \lambda} = \frac{\displaystyle\sum_{k=0}^{K-1} \sum_{j=1}^{m_k} \gamma^k \lambda^{j-1}}{K \gamma^{K-1}}. \tag{G.29}$$

For $m_{\max} = 1$ and $\gamma = 1$, this gives $\kappa_{K,1,1,\lambda} = 1$. With multi-step branching, the theorem uses only the fraction of the normalized objective assigned to one-step targets, which explains the dependence on $m_{\max}$ and λ.

Remark G.3 (Why reference-state SFT requires coverage)**.** Standard SFT controls the one-step error under solver-generated reference states:

$$\mathcal{E}_{\mathrm{SFT}}(G) := \frac{1}{K}\sum_{k=0}^{K-1} \mathbb{E}_{u\sim d_S^k}\left[\ell\left(G(u), S(u)\right)\right], \tag{G.30}$$

where d_S^k is the distribution of u_k^{ref}. To convert this quantity into an error under the deployment distribution d_G^k, one needs an absolute-continuity condition. Define

$$C(G) := \max_{0\le k<K} \left\| \frac{\mathrm{d}d_G^k}{\mathrm{d}d_S^k} \right\|_{L^\infty(d_S^k)}, \tag{G.31}$$

with $C(G) = +\infty$ if d_G^k is not absolutely continuous with respect to d_S^k for some k. A change of measure gives

$$\mathcal{E}_{\mathrm{on}}(G) \le C(G)\mathcal{E}_{\mathrm{SFT}}(G). \tag{G.32}$$

Consequently, an SFT-based rollout guarantee necessarily contains the additional factor $C(G)$:

$$\mathcal{E}_{\mathrm{SaT}}(G;S) \le \left(\sum_{r=0}^{K-1} L_S^r\right)\sqrt{\frac{C(G)\mathcal{E}_{\mathrm{SFT}}(G)}{\beta_1}}. \tag{G.33}$$

In deterministic systems with a fixed initial condition, once the student and reference trajectories reach different states, d_G^k and d_S^k may be distinct Dirac measures, in which case $C(G) = +\infty$. Thus, a small SFT loss on reference states alone need not imply a small deployment error.

Remark G.4 (Why SaT-1 does not provide solver correction)**.** SaT-1 rolls out the student but compares its prediction against the time-aligned reference target u_{k+1}^{ref}. The locally correct solver target at the student-visited state is instead $S(\hat{u}_k)$. These two targets differ whenever the student has departed from the reference trajectory. Indeed,

$$\begin{aligned}\left\|G(\hat{u}_k) - S(\hat{u}_k)\right\|_{L^2(\Omega)} &\le \left\|G(\hat{u}_k) - u_{k+1}^{\mathrm{ref}}\right\|_{L^2(\Omega)} + \left\|u_{k+1}^{\mathrm{ref}} - S(\hat{u}_k)\right\|_{L^2(\Omega)} \\ &\le \left\|G(\hat{u}_k) - u_{k+1}^{\mathrm{ref}}\right\|_{L^2(\Omega)} + L_S\left\|\hat{u}_k - u_k^{\mathrm{ref}}\right\|_{L^2(\Omega)}.\end{aligned} \tag{G.34}$$

Therefore, even if the time-aligned anchor loss is small, an additional state-distribution mismatch term remains. SaT-0 and SaT-.5 remove this ambiguity through the explicit target $S(\hat{u}_k)$.

Remark G.5 (Scope of the no-regret argument)**.** The online-to-batch part of Theorem G.1 treats the rollout states $\hat{u}_{n,k}$ as fixed after they are collected. It therefore corresponds most directly to the DAgger/detach implementation, in which the states are generated without gradient and then independently re-forwarded for optimization. The BPTT implementation uses the same solver targets and the same objective values, but additionally differentiates through the state-generation process. Establishing a no-regret result for that parameter-dependent state distribution would require a separate nonconvex optimization analysis.

The theorem is consequently conditional on the realized regret R_N. When $R_N = o(N)$, the online optimization contribution R_N/N vanishes, leaving only the best-in-class approximation error ε_N^α.

Remark G.6 (Horizon dependence and the DAgger interpretation)**.** The rollout amplification factor is

$$A_K(L_S) := \sum_{r=0}^{K-1} L_S^r = \begin{cases} \dfrac{1-L_S^K}{1-L_S}, & L_S \ne 1, \\ K, & L_S = 1. \end{cases} \tag{G.35}$$

When $L_S < 1$, local errors contract and $A_K(L_S)$ is uniformly bounded in K. When $L_S = 1$, the worst-case dependence is linear, whereas $L_S > 1$ permits geometric error amplification, as expected in unstable or chaotic regimes.

In the classical bounded-error DAgger interpretation [50], SFT controls a per-step error ϵ only under the expert-state distribution. The probability of having departed from that distribution by step k can then scale as $O(k\epsilon)$, producing $O(K^2\epsilon)$ cumulative error. On-policy data aggregation instead controls the error on learner-visited states and gives $O(K\epsilon)$ cumulative error when the expert can recover from local deviations. The continuous-state result above is the analogous statement for the SaT loss: distribution alignment is captured by $\mathcal{E}_{\mathrm{on}}$, while error propagation is captured by $A_K(L_S)$.

H Higher Resolution: 96^3

We retrained DPOT-Ti and P3D-S with SaT-1 and SaT-0 on all four 3D tasks at 96^3, one run per cell, with full fine-tuning, learning rate 10^{-3}, a plateau scheduler, $K = 5$, $\gamma = \lambda = 0.9$, effective batch size 48, and 160 epochs including a 40-epoch single-step warm-up, on NVIDIA H100 GPUs. Data follow the 64^3 setup except for the grid size. Table 17 reports cross-model 50-step RMSE (mean $\pm$ std over the two backbones) in units of 10^{-2}.

Table 17: Cross-model 50-step RMSE at 96^3 ($\times 10^{-2}$; DPOT-Ti and P3D-S).

Task	SaT-1	SaT-0
Burgers3D	1.27 ± 0.49	$\mathbf{1.11 \pm 0.40}$
KF3D	12.97 ± 3.25	$\mathbf{12.46 \pm 2.90}$
KS3D	143.67 ± 113.91	$\mathbf{21.65 \pm 7.46}$
DT3D	4.06 ± 1.06	$\mathbf{2.95 \pm 0.01}$

Clearly, SaT-0 improves on SaT-1 on all four tasks at 96^3, as at 64^3.

I Static Solver Data versus On-Policy Targets

To separate the effect of more solver data from that of querying the solver at student-visited states, we retrained SaT-1 on an extended static dataset generated by the same solver from fresh initial conditions, with the same number of targets per epoch as SaT-0 (SaT-1-Ext). All other columns are the values from table 22. Table 18 reports 50-step RMSE in units of 10^{-2}.

Table 18: Extended static data (SaT-1-Ext) against on-policy targets, 50-step RMSE ($\times 10^{-2}$). Best per row **bolded**.

Task	Model	Pretrained	SFT	SaT-1	SaT-1-Ext	SaT-.5	SaT-0
Burgers2D	DPOT-Ti	245.02	1.26	1.45	1.20	**0.66**	1.10
KS2D	DPOT-Ti	138.53	6.07	6.00	5.54	**4.39**	5.49
KS3D	DPOT-Ti	47.35	17.59	18.44	18.19†	17.92	**16.77**
KF3D	DPOT-Ti	44.58	16.50	12.41	11.49†	11.62	**10.99**
KS3D	P3D-S	235865.41	624.59	44.65	29.70	**25.34**	27.52
KF3D	P3D-S	77721.85	175.30	15.18	15.00	14.95	**14.78**

More static data improves SaT-1 in all six cells, by 1.2% to 33.5%, but an on-policy variant remains best in every row, by 1.5% to 45.0% over SaT-1-Ext. The best on-policy variant remains lower than both SaT-1 and SaT-1-Ext (25.34 vs. 29.70 on KS3D; 14.78 vs. 15.00 on KF3D).

J On-Policy Distillation with Task-Specific Learned Teachers

OPD in the main text uses a large fine-tuned foundation model as teacher. To test whether a teacher that matches the task's inductive bias closes the gap, we replaced it with a task-specific operator trained from scratch, an FNO in 2D and an AFNO in 3D (OPD-FNO/AFNO), and kept the student protocol unchanged. OPD and SaT-0 are the values from table 1. Table 19 reports cross-model 50-step RMSE in units of 10^{-2}.

Although these operators match the periodic, pseudo-spectral setting, students distilled from them are $1.14\times$ to $2.47\times$ worse than students distilled from the large teacher, and SaT-0 is lower than both learned teachers on all seven tasks. A learned operator is fitted on the data manifold and need not remain accurate after the student drifts, whereas a numerical solver stays consistent with its governing equation and discretization.

Table 19: OPD with a large FM teacher, OPD with a task-specific FNO/AFNO teacher, and SaT-0; cross-model 50-step RMSE ($\times 10^{-2}$).

Task	OPD	OPD-FNO/AFNO	SaT-0
Burgers2D	1.93 ± 0.96	2.27 ± 1.51	$\mathbf{1.18 \pm 0.40}$
KS2D	10.2 ± 4.22	15.2 ± 2.34	$\mathbf{8.41 \pm 4.19}$
KF2D	78.5 ± 28.3	89.3 ± 37.7	$\mathbf{54.1 \pm 20.9}$
Burgers3D	1.09 ± 0.12	2.21 ± 0.07	$\mathbf{0.80 \pm 0.14}$
KS3D	37.7 ± 20.8	93.1 ± 93.5	$\mathbf{22.4 \pm 4.40}$
KF3D	14.2 ± 1.43	21.5 ± 2.50	$\mathbf{12.9 \pm 1.55}$
DT3D	3.98 ± 1.71	5.68 ± 0.72	$\mathbf{3.04 \pm 0.91}$

K On-Policy Physics-Informed Baseline

To test whether on-policy sampling with PDE residuals suffices, we post-trained the 2D students with a PINO-style residual loss [51] evaluated at student-visited states, without solver targets (PINO-OP-0), and with an additional reference-trajectory anchor of weight 0.5 (PINO-OP-.5). SaT-0 is the value from table 1. Table 20 reports cross-model 50-step RMSE in units of 10^{-2}.

Table 20: On-policy residual baselines and SaT-0; cross-model 50-step RMSE ($\times 10^{-2}$).

Task	PINO-OP-0	PINO-OP-.5	SaT-0
Burgers2D	11.46 ± 7.27	3.01 ± 2.47	$\mathbf{1.18 \pm 0.40}$
KS2D	33.55 ± 7.73	16.14 ± 5.78	$\mathbf{8.41 \pm 4.19}$
KF2D	181.31 ± 31.06	90.91 ± 29.48	$\mathbf{54.09 \pm 20.89}$

Clearly, SaT-0 substantially outperforms both PINO-OP-0 and PINO-OP-.5 across all three reported tasks. We attribute the gap between SaT and PINO-OP to two complementary properties of SaT's multi-step targets. First, SaT supervises the state directly, whereas PINO-OP optimizes a residual objective, which is harder to optimize: adding a reference-trajectory anchor reduces the error of PINO-OP by 49.9% to 73.7%, and SaT-0 reduces it by a further 40.5% to 60.8% by using solver-derived targets at student-visited states. Second, and most importantly, the targets are on-policy and finite-horizon. SaT matches the student's j-step prediction from $\hat{u}_k$ to the solver output $\mathcal{S}^j(\hat{u}_k)$ for several horizons j. Both sides of each loss term $\ell(\hat{u}_{k+j}, \mathcal{S}^j(\hat{u}_k))$ start from the same student-visited state, so each term isolates the error introduced within its window and penalizes it at the magnitude it reaches after propagating through the intervening dynamics. PINO-OP instead constrains each visited state through a local residual, which measures an error only at its instantaneous size. The difference matters most in chaotic regimes, where errors along unstable directions grow rapidly. This comparison covers the three 2D tasks; without an anchor, PINO-OP-0 is poorly conditioned on these chaotic systems, which makes the comparison favorable to SaT.

L Wall-Bounded Turbulent Channel Flow with a Finite-Volume Solver

TCF3D is turbulent channel flow at $\mathrm{Re}_\tau = 180$ from FluidGym [52], on a $64 \times 65 \times 64$ grid refined toward the walls, with no-slip walls in the wall-normal direction and periodic boundaries in the streamwise and spanwise directions. The state has four channels (u_x, u_y, u_z, p), and one step advances $\Delta t = 0.005$. The teacher is PICT [48], a PyTorch finite-volume solver with an implicit PISO pressure projection and an iterative linear solve at every step. It shares no component with Exponax, and SaT uses it through the same solver interface without changes. Pressure enforces incompressibility rather than evolving as a state, so the post-training losses and all reported metrics use the three velocity channels.

Each run uses a 40-epoch single-step warm-up and 160 main epochs, full fine-tuning, learning rate 10^{-3} with a cosine schedule, and effective batch size 32. The checkpoint is selected by 50-step RMSE on a held-out validation set. OPD uses P3D-L, fine-tuned on TCF3D, as its teacher, with a batch size of 6. We report the best run per model and method in table 21.

Table 21: TCF3D velocity RMSE after 50 steps. Best per row **bolded**.

Model	SFT	OPD	SaT-1	SaT-.5	SaT-0
DPOT-Ti	0.4431	0.5252	0.4162	0.4138	**0.4092**
MORPH-Ti	0.6402	0.5776	0.5253	0.5192	**0.5060**
P3D-S	21.4909	0.8161	0.1388	**0.0863**	0.1022
Mean $\pm$ std	7.5247 ± 9.8759	0.6396 ± 0.1266	0.3601 ± 0.1627	0.3398 ± 0.1843	$\mathbf{0.3391 \pm 0.1721}$

A SaT variant gives the lowest velocity RMSE on all three backbones (SaT-0 on DPOT-Ti and MORPH-Ti, SaT-.5 on P3D-S), and SaT-0 the lowest cross-model RMSE; SFT diverges on P3D-S. Figure 8 shows the P3D-S velocity field of every method after 50 steps.

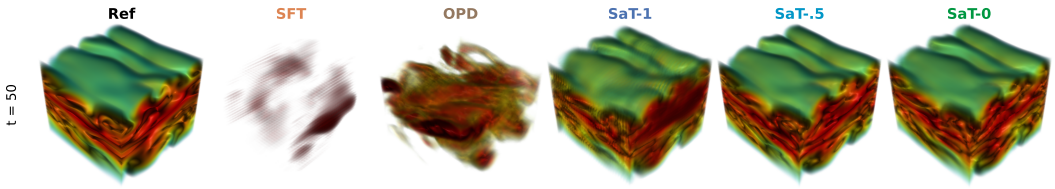


Figure 8: TCF3D, velocity magnitude at $t = 50$ (P3D-S). SFT has diverged and retains no flow structure. OPD loses the layered channel-flow profile. All three SaT variants keep the layered profile and the near-wall streaks: SaT-1 blurs the streaks and smears the small-scale eddies, while SaT-.5 and SaT-0 remain closest to the reference, consistent with table 21.

M Detailed In-Distribution Per-Model RMSE

Table 22 reports the per-model 50-step rollout RMSE that is aggregated across architectures in the main text (table 1). Bold values indicate the best method per model within each task; bold underlined values indicate the best across all models for that task.

N Detailed OOD Zero-Shot Results

Table 23 reports per-model RMSE for all OOD tasks. The main text (Table 2) aggregates these results as mean $\pm$ std across four 2D architectures or three 3D architectures; here we provide the full per-model breakdown. Bold values indicate the best method per model within each task; bold underlined values indicate the best across all models for that task.

Table 22: Per-model rollout RMSE on 2D and 3D in-distribution tasks across 7 PDE benchmarks and 6 foundation models. Best per model **bolded**; best per task **underlined**. PITA does not support 3D tasks (–).

Task	Model	Pretrained	SFT	PITA	OPD	SaT-1	SaT-.5	SaT-0
Burgers2D	DPOT-Ti	2.4502	0.0126	0.0103	0.0160	0.0145	**0.0066**	0.0110
	PDE-S	0.6937	0.0446	0.0515	0.0347	0.0150	0.0287	**0.0120**
	Poseidon-T	0.3508	0.0103	0.0086	0.0084	0.0078	**0.0042**	0.0063
	MPP-B	0.2760	0.0838	0.0222	0.0180	0.0181	**0.0170**	0.0176
KF2D	DPOT-Ti	2.7181	0.2316	0.3674	0.4018	0.2340	0.2966	**0.1988**
	PDE-S	2.3416	1.0938	1.0617	1.1566	0.7190	1.1282	**0.6562**
	Poseidon-T	2.2831	0.8494	0.9367	0.6581	0.5699	0.6701	**0.5583**
	MPP-B	3.0967	1.4343	0.8410	0.9215	0.8307	0.9265	**0.7504**
KS2D	DPOT-Ti	1.3853	0.0607	0.0566	0.0716	0.0600	**0.0439**	0.0549
	PDE-S	0.3646	0.1341	0.1757	0.1415	0.1219	0.1307	**0.1089**
	Poseidon-T	0.4196	0.0397	0.0531	0.0489	0.0344	**0.0278**	0.0335
	MPP-B	0.3358	0.2401	0.1484	0.1445	0.1489	**0.1351**	0.1389
Burgers3D	DPOT-Ti	0.1435	0.0103	–	0.0125	0.0099	0.0109	**0.0094**
	MORPH-Ti	1.2382	0.0154	–	0.0108	**0.0084**	0.0095	0.0085
	P3D-S	2251.4734	0.0206	–	0.0095	0.0064	0.0070	**0.0061**
KS3D	DPOT-Ti	0.4735	0.1759	–	0.2012	0.1844	0.1792	**0.1677**
	MORPH-Ti	1.3442	0.2352	–	0.2595	0.2369	0.2306	**0.2285**
	P3D-S	2358.6541	6.2459	–	0.6696	0.4465	**0.2534**	0.2752
DT3D	DPOT-Ti	0.1975	0.0557	–	0.0638	0.0529	0.0570	**0.0371**
	MORPH-Ti	0.8630	0.0418	–	**0.0308**	0.0362	0.0335	0.0366
	P3D-S	972.1893	0.6687	–	0.0248	0.0184	0.0217	**0.0175**
KF3D	DPOT-Ti	0.4458	0.1650	–	0.1222	0.1241	0.1162	**0.1099**
	MORPH-Ti	0.9374	0.1780	–	0.1550	0.1338	0.1326	**0.1299**
	P3D-S	777.2185	1.7530	–	0.1497	0.1518	0.1495	**0.1478**

Table 23: Per-model RMSE on 2D and 3D OOD zero-shot tasks. Best per model **bolded**; best per task **underlined**. PITA does not support 3D tasks (–).

Task	Model	SFT	PITA	OPD	SaT-1	SaT-.5	SaT-0
Burgers2D-OOD	DPOT-Ti	0.0746	0.0713	0.0733	0.0728	**0.0708**	0.0723
	PDE-S	0.0843	0.0905	0.0748	**0.0725**	**0.0725**	**0.0725**
	Poseidon-T	0.0719	0.0717	0.0715	0.0707	**0.0706**	0.0707
	MPP-B	0.1115	0.0746	0.0758	0.0746	0.0747	**0.0735**
KS2D-OOD	DPOT-Ti	0.0888	0.0898	0.0889	0.0937	**0.0851**	0.0853
	PDE-S	0.1462	0.1790	0.1517	0.1748	0.1498	**0.1335**
	Poseidon-T	0.1395	0.1023	0.1552	0.1246	**0.0865**	0.0995
	MPP-B	0.1746	0.1088	0.1090	0.1071	0.1061	**0.1059**
KF2D-OOD	DPOT-Ti	2.0029	2.1528	2.1561	2.0670	2.0947	**1.9956**
	PDE-S	2.6950	2.7148	2.6940	**2.3248**	2.7282	2.3858
	Poseidon-T	2.3918	2.3602	2.2884	2.3838	2.4473	**2.1338**
	MPP-B	2.7200	2.6462	2.5519	2.5508	2.7253	**2.5457**
Burgers3D-OOD	DPOT-Ti	0.0154	–	0.0171	0.0147	0.0168	**0.0143**
	MORPH-Ti	0.0161	–	0.0130	0.0127	0.0133	**0.0125**
	P3D-S	0.0208	–	0.0127	**0.0119**	0.0121	0.0120
KF3D-OOD	DPOT-Ti	0.2234	–	0.1964	0.1933	0.1895	**0.1866**
	MORPH-Ti	0.2455	–	0.2097	0.1994	0.1989	**0.1955**
	P3D-S	2.7287	–	0.2001	0.2037	0.2006	**0.1967**
KS3D-OOD	DPOT-Ti	0.1748	–	0.2031	0.1763	0.1724	**0.1614**
	MORPH-Ti	0.2161	–	0.2424	0.2221	**0.2117**	0.2155
	P3D-S	7.6484	–	0.4237	0.2029	0.2027	**0.1990**
DT3D-OOD	DPOT-Ti	0.0533	–	0.0594	0.0514	0.0540	**0.0404**
	MORPH-Ti	0.0428	–	0.0338	0.0379	0.0381	**0.0337**
	P3D-S	0.6499	–	0.0371	0.0383	0.0376	**0.0368**

O Full 3D Trajectory Visualizations

We visualize full 50-step autoregressive rollouts on the four 3D PDE tasks at 64^3 resolution. Each figure shows volumetric iso-surface snapshots at steps $t \in \{1, 10, 20, 30, 40, 50\}$ (rows), comparing the solver reference, the pretrained backbone, SFT, OPD, and SaT-0 (columns). These complement the quantitative RMSE results in Tables 22 and 23 by exposing how field morphology degrades over time for each training method.

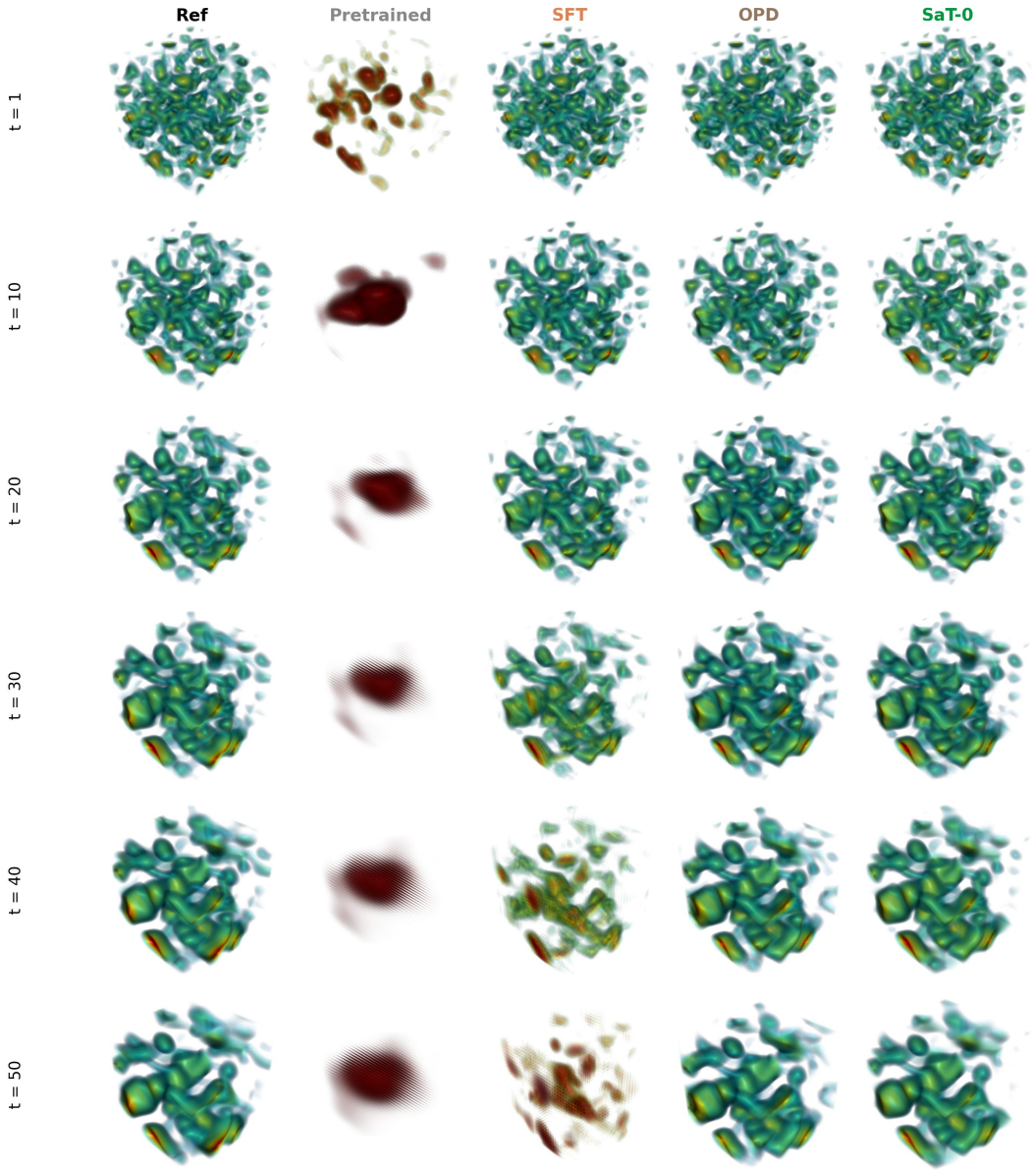


Figure 9: Burgers3D (P3D-S, trajectory 0). The pretrained backbone collapses into a single large-scale blob by $t = 10$; SFT tracks the reference well until $t \approx 30$ before developing grid-level artifacts at $t = 40$–50. OPD and SaT-0 both preserve the fine-scale shock structure throughout the full 50-step rollout, with SaT-0 maintaining the closest visual match to the reference at late times.

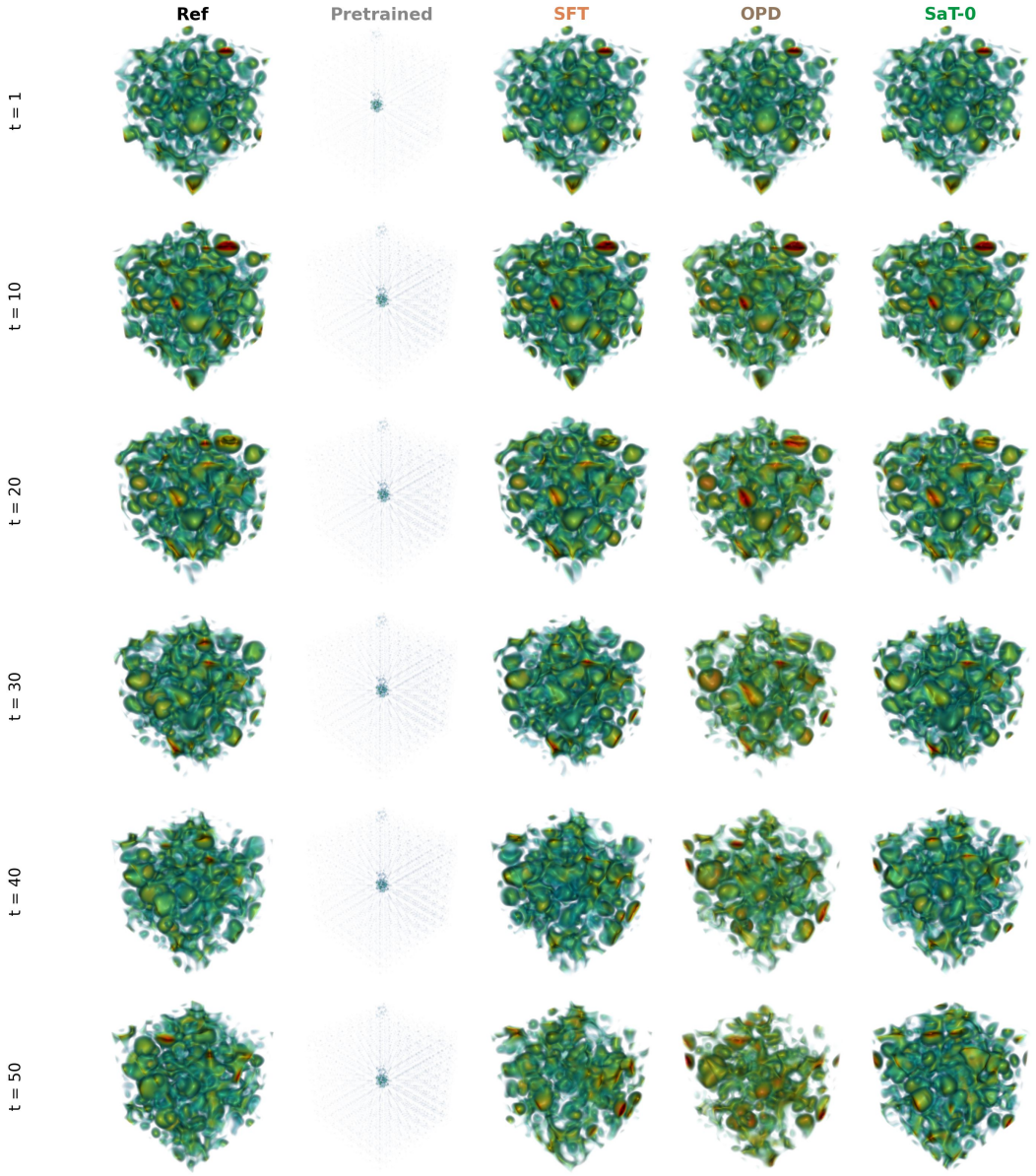


Figure 10: KS3D — Kuramoto–Sivashinsky (DPOT-Ti, trajectory 3). The chaotic dynamics of the KS equation make long-horizon tracking particularly challenging. The pretrained backbone diverges immediately. SFT maintains qualitative agreement until $t \approx 20$ but loses fine-scale cellular patterns at later steps. Both OPD and SaT-0 preserve the characteristic cellular structure throughout, with SaT-0 exhibiting the sharpest structures and most accurate magnitude at $t = 50$.

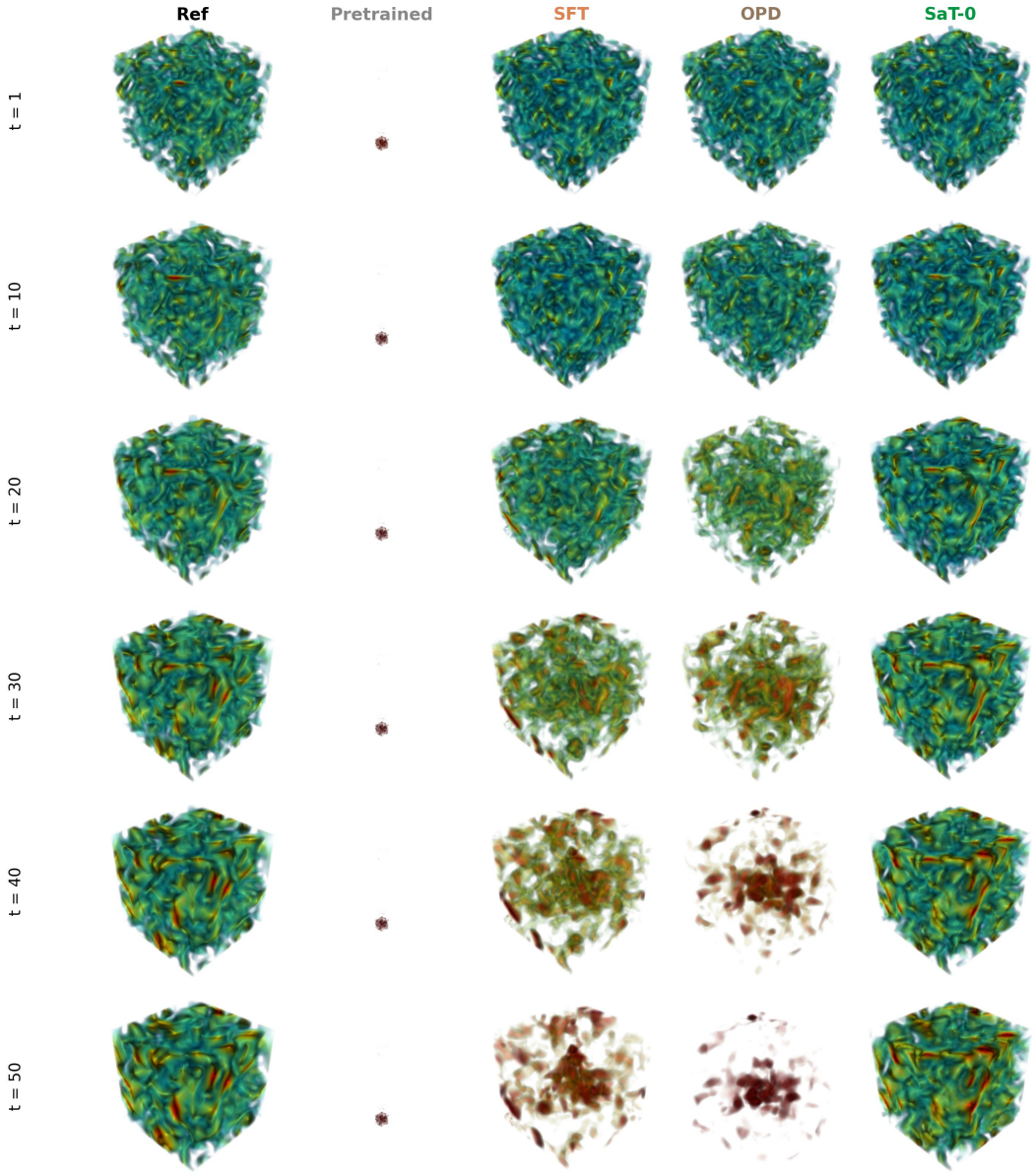


Figure 11: DT3D — Decaying Turbulence velocity (DPOT-Ti, trajectory 2). As kinetic energy decays, fine-scale vortices merge and dissipate. The pretrained model fails to reproduce this cascade. SFT captures the large-scale decay but over-smooths the remaining eddies. OPD and SaT-0 reproduce both the energy decay rate and the vortex morphology more faithfully; SaT-0 retains the sharpest eddy boundaries at late times.

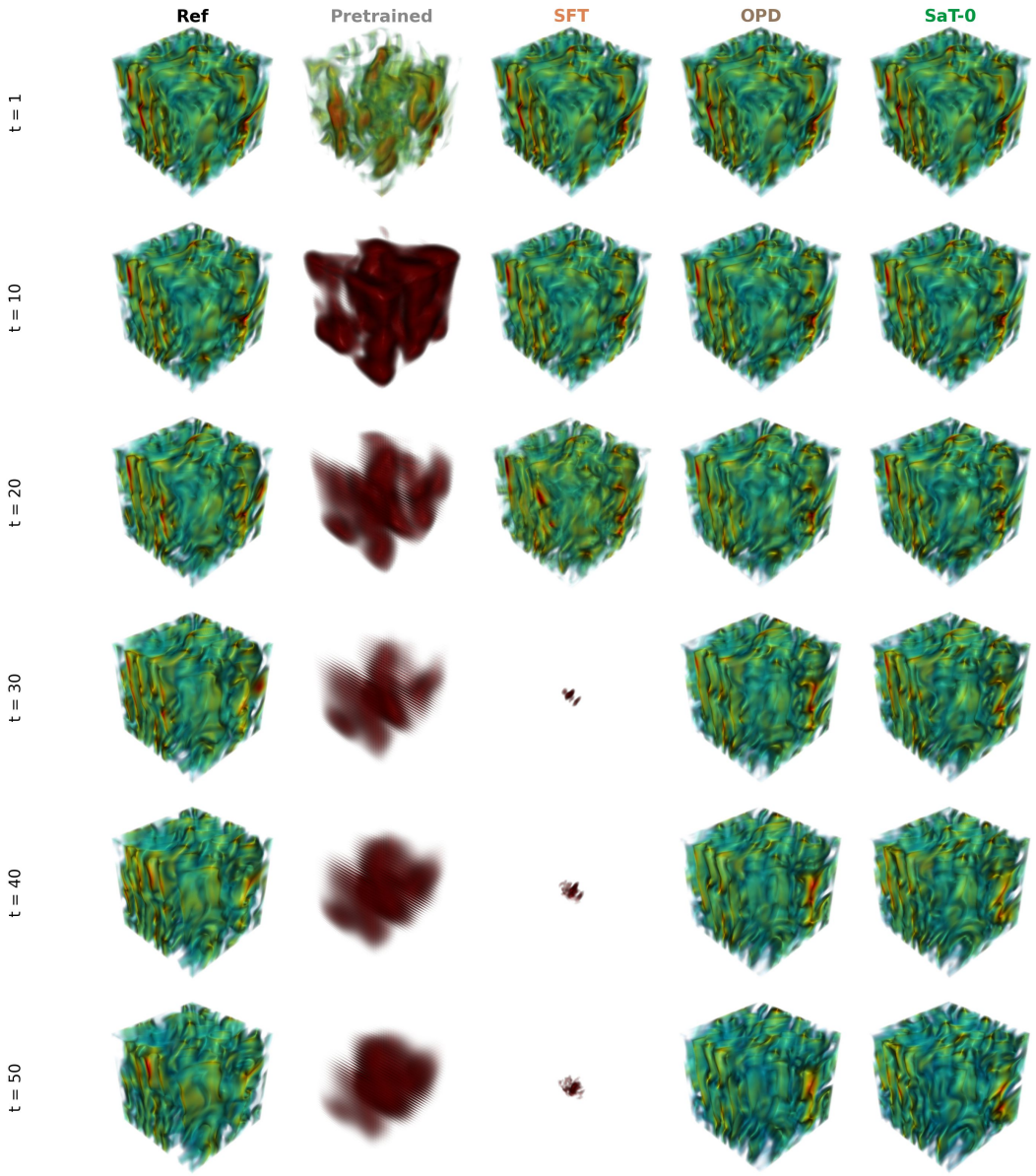


Figure 12: KF3D — Kolmogorov Flow velocity (P3D-S, trajectory 0). The pretrained model fails to produce physically meaningful velocity iso-surfaces. SFT captures the large-scale flow topology but progressively over-smooths fine vortical structures after $t = 20$. OPD and SaT-0 retain multi-scale vortex features up to $t = 50$; SaT-0 shows slightly sharper structures at late times.

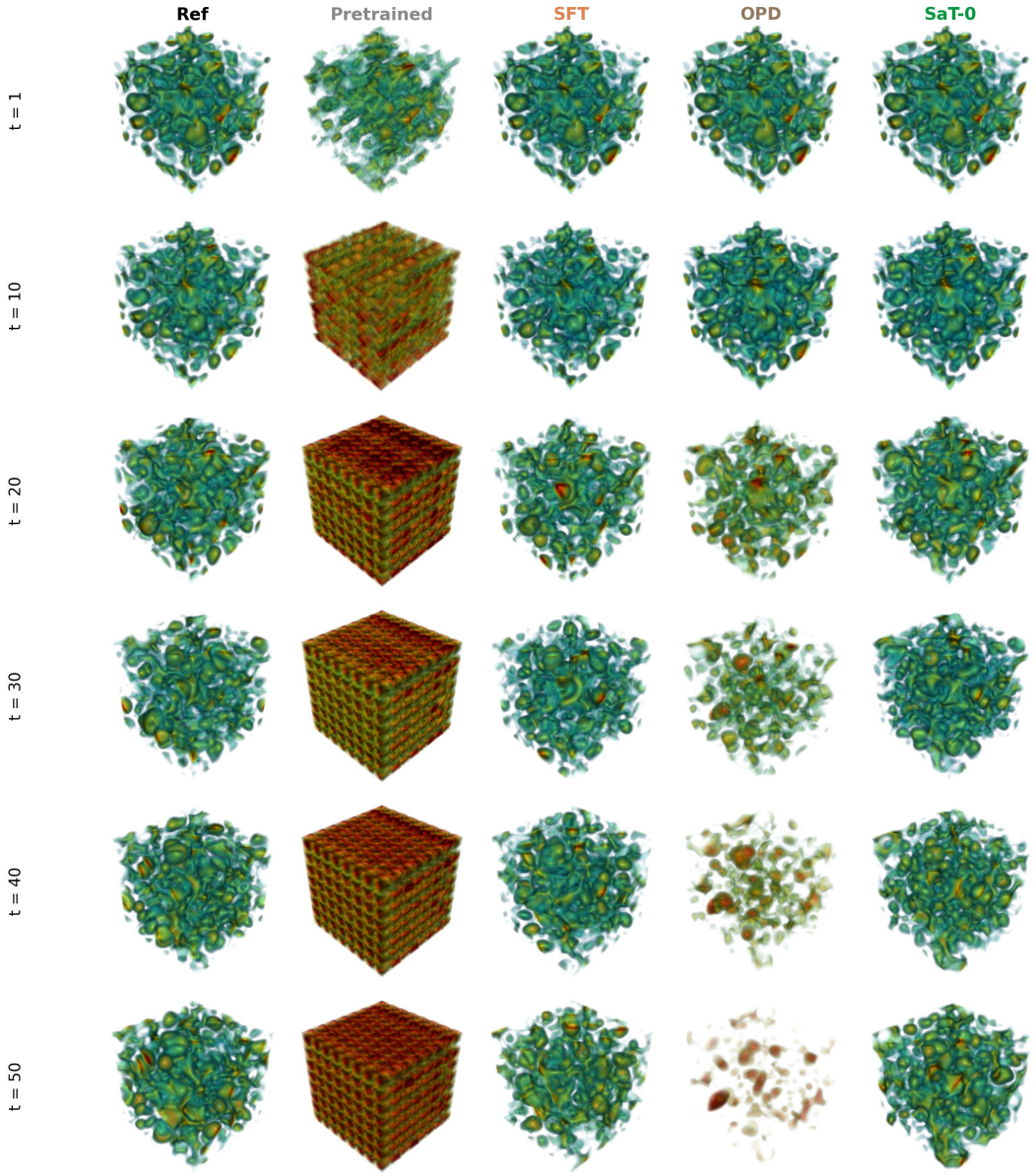


Figure 13: KS3D — Kuramoto–Sivashinsky (MORPH-Ti, trajectory 4). The chaotic dynamics make long-horizon tracking particularly challenging. The pretrained backbone diverges immediately. OPD maintains qualitative agreement at early steps but loses fine-scale cellular patterns at later steps. Both SFT and SaT-0 preserve the characteristic cellular structure throughout the full 50-step rollout, with SaT-0 showing sharper structures at late times.

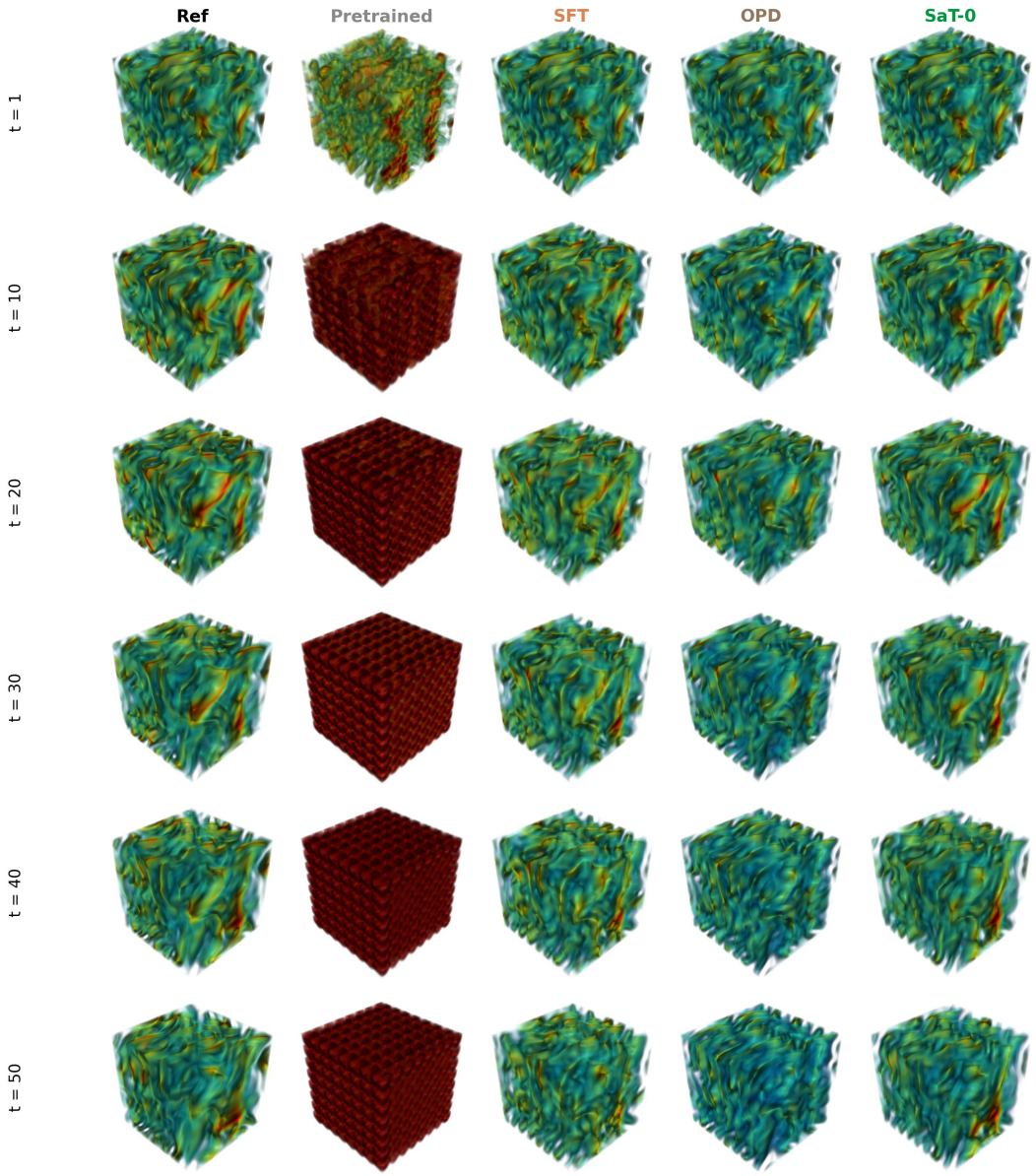


Figure 14: KF3D — Kolmogorov Flow velocity (MORPH-Ti, trajectory 4). The pretrained model fails to produce physically meaningful velocity iso-surfaces. OPD and SFT capture the large-scale flow topology but progressively over-smooths fine vortical structures after $t = 20$. SaT-0 retains multi-scale vortex features up to $t = 50$ and shows sharper structures at late times.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The abstract and introduction (section 1) state three contributions: (1) the SaT on-policy distillation framework, described in section 3; (2) the unified post-training suite, detailed in section 4; and (3) the empirical evaluation across seven PDE benchmarks and six backbones, reported in section 5. All quantitative claims (23 of 24 best in-distribution, up to 90.0% RMSE reduction, best on all 7 OOD tasks) are substantiated by the tables and figures in section 5.

2. **Limitations**

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

   Answer: [Yes]

   Justification: section 6 contains an explicit "Limitations" paragraph discussing three key limitations: (1) SaT requires access to a reliable numerical solver at training time; (2) the empirical evaluation does not yet extend to complex geometries or unstructured meshes; and (3) SaT's efficiency depends on successful solver parallelization.

3. **Theory assumptions and proofs**

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

   Answer: [Yes]

   Justification: appendix G states the assumptions of theorem G.1 and gives its complete proof.

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: The method is fully described in section 3. section 4 specifies all PDE benchmarks, model architectures, and the teacher solver. The appendix provides complete experiment details: governing equations and solver parameters (appendix E), model architecture descriptions (appendix D), training strategy and hyperparameter grid (appendix E.5), OOD dataset specifications (appendix E.4), and metric definitions (appendix F).

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

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

   Answer: [Yes]

   Justification: The source code and training configurations will be released upon publication. The 2D benchmark data are publicly available through APEBench, and the 3D datasets are generated using the open-source Exponax solver with parameters fully specified in appendix E.3 and will also be released upon publication.

6. **Experimental setting/details**

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

   Answer: [Yes]

Justification: section 4 describes the models, baselines, and PDE benchmarks. The appendix provides complete details: data splits and dataset generation (appendix E.3), the hyperparameter grid and model selection procedure (appendix E.5), optimizer settings (AdamW, gradient clipping, learning rate), and per-model architecture configurations (appendix D).

7. **Experiment statistical significance**

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

   Answer: [Yes]

   Justification: The cross-model aggregate tables (tables 1, 2 and 16) report mean $\pm$ one standard deviation across evaluated architectures (4 for 2D, 3 for 3D), capturing variability across model families. The per-model breakdown tables (tables 22 and 23) complement these with individual model results. The bar-chart figures (figs. 2 and 5) include error bars.

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: All training runs use a single NVIDIA A100 GPU (appendix E.5). Per-iteration timing breakdowns are provided in appendix A (Tables 3–4, profiled on an NVIDIA L40S GPU). End-to-end training wall-clock times for all model–method pairs are reported in appendix A.1 (Table 6).

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: We have read and commit to adhering to the NeurIPS Code of Ethics. The work uses only synthetic PDE simulation data and publicly available pretrained models, and poses no risks to human subjects, privacy, or fairness.

10. **Broader impacts**

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

    Answer: [N/A]

    Justification: This work is foundational research on post-training methods for PDE surrogate models. It operates on synthetic physical simulation data and does not have direct negative societal applications. Positive impacts (accelerating scientific computing) are implicit in the problem motivation.

11. **Safeguards**

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

    Answer: [N/A]

    Justification: The paper poses no such risks. The released assets are post-training code and fine-tuned PDE surrogate model checkpoints, which are domain-specific scientific simulation tools with no foreseeable misuse potential.

12. **Licenses for existing assets**

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

    Answer: [Yes]

Justification: All pretrained PDE foundation models are cited with their original publications (section 4 and appendix D). The APEBench benchmark suite and Exponax solver are cited in appendix E. All assets used are publicly released under open-source licenses by their respective authors.

13. **New assets**

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

    Answer: [Yes]

    Justification: The paper introduces a unified post-training framework (code) and fine-tuned model checkpoints. The training pipeline, configuration system, and all hyperparameters are documented in sections 3 and 4 and the appendix. The code will be available at https://github.com/tum-pbs/Solver-as-Teacher.git.

14. **Crowdsourcing and research with human subjects**

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

    Answer: [N/A]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

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

    Answer: [N/A]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

16. **Declaration of LLM usage**

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

    Answer: [N/A]

    Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components. During the rebuttal stage, the authors used an LLM coding assistant (Claude Opus 5.5) as a software tool to monitor cluster jobs, write analysis and plotting scripts, and help with editing.