HYBRIDOSS: ORTHOGONAL SCALE SPLITTING FOR HYBRID DIFFERENTIABLE PDE SOLVERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Fine-resolution simulation of partial differential equations (PDEs) is important for resolving small-scale structures and cross-scale interactions, but becomes computationally expensive over long time horizons. Neural surrogates reduce this cost but can suffer from error accumulation, while hybrid models typically require the numerical solver and corrected state to operate at the same spatial resolution. We propose HybridOSS, an orthogonal scale splitting framework that decouples numerical-solver resolution from state resolution. HybridOSS decomposes the fine-resolution state into complementary resolved and unresolved components. A differentiable numerical solver advances the resolved state on a coarse grid, while a closure module corrects the resolved dynamics and a refinement module explicitly evolves the unresolved state. Complementary spectral projections constrain the learned updates to orthogonal subspaces while preserving cross-scale coupling. Our instantiated HybridOSS model reduces global prediction error by 31% and 69% over the strongest hybrid baseline on KS and Allen–Cahn, respectively, while improving both resolved- and unresolved-scale accuracy. On differentiable OpenFOAM for 3D incompressible Navier–Stokes turbulence, HybridOSS maintains lower error over long autoregressive rollouts and nearby unseen viscosity values while achieving approximately a $19\times$ inference speedup over the fine-grid solver.

1 INTRODUCTION

Partial differential equations (PDEs) describe many systems in science and engineering. In many applications, the quantities of interest depend on fine-scale structures and cross-scale interactions that cannot be represented adequately on coarse grids (Moin & Mahesh, 1998; Piomelli, 1999). Preserving these dynamics therefore requires fine-resolution prediction over long time horizons. However, resolving such states in large-scale simulations with numerical solvers demands fine spatial grids and repeated time integration, making long-horizon simulation computationally expensive. (Alfonsi, 2009; Lee & Moser, 2015). Neural surrogates offer an alternative by learning to predict the evolving state from data (Chen et al., 2022; Liu et al., 2024). However, prediction errors can accumulate during autoregressive rollout, and accuracy may decline outside the training distribution (Mikhaeil et al., 2022; Jiang et al., 2023; List et al., 2025).

Hybrid differentiable models retain a numerical solver as a physics-based backbone and learn corrections from high-fidelity data to improve stability and accuracy (Um et al., 2020; Kochkov et al., 2021; Wei et al., 2025; List et al., 2025; Mezidi et al., 2026). However, existing formulations typically evolve the numerical solution and learned correction at the same spatial resolution. Fine-resolution prediction therefore still requires a fine-grid numerical backbone, while reducing the solver resolution naturally restricts the evolved state to a coarse representation.

To overcome this resolution bottleneck, we consider using a coarse numerical solver while maintaining a fine-resolution state. Projecting the PDE onto complementary frequency subspaces naturally separates the dynamics into two roles. The unresolved scales influence the resolved dynamics through a closure term, while the unresolved component itself evolves over time (Devulder & Marion, 1992; Piomelli, 1999; Gouasmi et al., 2017). This separation therefore provides a natural inductive bias for learned correction. Accordingly, we assign resolved-scale correction and unresolved-state evolution to separate modules and constrain their updates to the corresponding subspaces.

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

Our approach. We propose HybridOSS, an orthogonal scale splitting framework that decouples numerical-solver resolution from state resolution. The state is decomposed as $u = u_{\mathrm{lo}} + u_{\mathrm{hi}}$, where the two components occupy complementary frequency subspaces. A differentiable numerical solver advances the resolved state on a coarse grid, while a closure module learns the residual correction required in the resolved subspace, including effects the feedback of unresolved scales. Meanwhile, A refinement module evolves the unresolved state on the fine grid. At the same time, both modules receive information from the coupled state, so the orthogonal output constraint does not decouple the underlying dynamics. The resulting architecture combines numerical evolution at a reduced spatial resolution with explicit learned evolution of the unresolved state. Our main contributions can be summarized as follows:

- We formulate spectral coarsening as an orthogonal decomposition that provides a reduced state to the coarse-grid solver while retaining the complementary unresolved component as an explicit dynamical state.
- We design HybridOSS around the OSS decomposition, combining coarse-grid numerical evolution with separate closure and refinement modules for resolved-scale correction and unresolved-state evolution. Complementary projections constrain their updates to orthogonal subspaces while preserving cross-scale coupling.
- HybridOSS demonstrates improved accuracy, efficiency, and stability on canonical PDE benchmarks, while further exhibiting strong long-rollout performance, generalization, and computational efficiency on a differentiable OpenFOAM turbulence case.

2 Related Work

Neural surrogate models. Neural operators provide a framework for constructing PDE surrogates by learning mappings between function spaces (Kovachki et al., 2023). Representative architectures include DeepONet's branch–trunk representation (Lu et al., 2021), graph-based kernel aggregation in GNO (Li et al., 2020), and Fourier-space transformations in FNO (Li et al., 2021a). Attention-based surrogates offer another way to model interactions among spatial locations, but applying dense attention to all grid points incurs quadratic cost in their number. To address this cost, existing methods use efficient attention formulations (Li et al., 2022; 2023; Hao et al., 2023; Hu et al., 2026) or aggregate spatial information into fewer latent tokens (Wang & Wang, 2024; Alkin et al., 2024) or learned physical slices (Wu et al., 2024; Luo et al., 2025; Zhou et al., 2026).

Although surrogate models can reduce the cost of individual prediction steps, accurate local predictions are hard to ensure reliable long-term evolution. Repeated application of a learned update can also shift the predicted state distribution away from the training data, compromising long-term statistics (Mikhaeil et al., 2022; Stachenfeld et al., 2022). Mitigation strategies include training over multiple rollout steps(Lam et al., 2023), perturbing training inputs (Sanchez-Gonzalez et al., 2020), and learning to preserve invariant measures (Pedersen et al., 2025; Li et al., 2021b).

Differentiable hybrid models. Hybrid PDE models combine numerical evolution with learned corrections. Solver-in-the-loop (SITL) training differentiates through successive solver steps so that the corrector is optimized in the context of the evolving trajectory (Um et al., 2020). Differentiable simulation frameworks support this integration (Holl & Thuerey, 2024), while learned numerical components have enabled accurate fluid simulation on coarse grids (Kochkov et al., 2021). INC studies indirect neural correction for autoregressive hybrid solvers (Wei et al., 2025), and Mezidi et al. (2026) use proximal splitting to organize the interaction between numerical and learned updates. These approaches motivate comparisons with different correction strategies, while analyses of unrolled training highlight the importance of differentiability and training design (List et al., 2025). These methods mainly differ in how learned correction interacts with numerical evolution, while the numerical state and corrected state typically remain at the same working resolution(Um et al., 2020; Kochkov et al., 2021; Wei et al., 2025; List et al., 2025; Mezidi et al., 2026). HybridOSS instead focuses on decoupling these resolutions within the SITL paradigm. It uses a coarse-grid numerical solver for the resolved state and assigns learned closure and refinement to complementary spectral subspaces, with refinement explicitly evolving the unresolved component. This allows the model to maintain fine-resolution dynamics over long-time simulation without requiring the numerical backbone to operate at the same fine resolution.

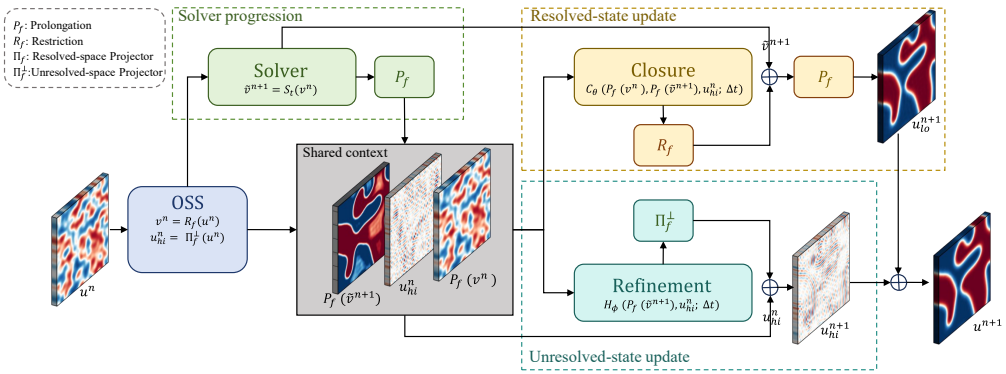


Figure 1: Overview of the HybridOSS framework.

Scale decomposition and closure modeling. Separating resolved and unresolved dynamics has a long history in numerical modeling. Variational multiscale and nonlinear Galerkin methods decompose PDE dynamics across complementary scales (Hughes et al., 1998; Bazilevs et al., 2007; Devulder & Marion, 1992), while large-eddy simulation models the influence of unresolved motions on resolved dynamics through closure (Piomelli, 1999). These works provide the conceptual basis for scale-aware modeling. Building on this perspective, HybridOSS retains the unresolved complement as an explicit dynamical state and assigns resolved-scale correction and unresolved-state evolution to separate learned updates over complementary spectral subspaces.

3 Method

HybridOSS decouples the spatial resolution of numerical evolution from that of the maintained state. Each fine-grid state is decomposed into complementary resolved and unresolved components. A numerical solver advances the resolved component on a coarse grid, a closure module corrects its resolved dynamics, and a refinement module evolves the unresolved component on the fine grid. The prolonged coarse state and the updated unresolved state are then recombined to form the next fine-resolution prediction, as illustrated in Fig. 1.

3.1 Problem Setup and Overview

We study the task of learning a high-fidelity trajectory from a given initial condition while retaining a low-fidelity numerical solver as the physical backbone. Let $u^0 \in \mathbb{R}^{B\times C\times N^d}$ denote the initial state of a discretized periodic PDE, where B is the batch size, C is the number of physical variables, N is the number of fine-grid points per spatial direction, and $d \in \{1,2,3\}$ is the spatial dimension. Given reference states $\{u_{\mathrm{HF}}^n\}_{n=0}^T$ sampled at discrete times $t_n = n\Delta t$, the objective is to learn a rollout whose predicted trajectory approximates the reference trajectory:

$$\widehat{u}^{n+1} = \mathcal{H}_\vartheta(\widehat{u}^n;\Delta t), \qquad \widehat{u}^0 = u^0, \qquad \widehat{\mathcal{U}}_\vartheta = \{\widehat{u}^n\}_{n=0}^T \approx \mathcal{U}_{\mathrm{HF}} = \{u_{\mathrm{HF}}^n\}_{n=0}^T. \tag{1}$$

Here, n indexes discrete time points and ϑ collects the learned parameters. Reference trajectories provide supervision, but reference states are not inputs during autoregressive rollout. At each time step from t_n to t_{n+1}, the model advances the current predicted state, so errors can accumulate across the trajectory.

3.2 Spectral Coarsening and Orthogonal Decomposition

We decompose each fine-grid state into a resolved component representable on a coarse grid and its complementary unresolved component. Let the fine grid contain N points along each spatial dimension, let $f \geq 1$ be an integer coarsening factor dividing N, and set $N_c = N/f \geq 2$. The coarsening factor f controls the solver resolution and the division of dynamics between numerical and learned evolution, inducing an accuracy–efficiency trade-off that we examine in Section 4.3.

We denote the fine- and coarse-grid discrete Fourier transforms by $\mathcal{F}_N$ and $\mathcal{F}_{N_c}$, respectively. Let T_f retain the Fourier modes supported by the coarse grid and let E_f embed a coarse-grid spectrum into the fine-grid spectrum by zero padding. By the Nyquist–Shannon sampling theorem, frequencies below the coarse-grid Nyquist limit remain representable after coarsening (Shannon, 1949). We therefore define the spectral restriction and prolongation operators as

$$R_f(u) = \left(\mathcal{F}_{N_c}^{-1} \circ T_f \circ \mathcal{F}_N\right)(u), \qquad P_f(v) = \left(\mathcal{F}_N^{-1} \circ E_f \circ \mathcal{F}_{N_c}\right)(v). \tag{2}$$

For $f > 1$, T_f retains modes $|k_q| < k_c$, with $k_c = \lfloor N_c/2 \rfloor$, while excluding the coarse Nyquist mode to preserve Hermitian symmetry. For $f = 1$, both operators reduce to the identity. We further define the resolved-space projector

$$\Pi_f = P_f \circ R_f \tag{3}$$

and its complementary unresolved-space projector

$$\Pi_f^{\perp} = I_f - \Pi_f, \tag{4}$$

where I_f denotes the identity operator on the fine-grid state space. The operator Π_f retains the modes representable on the coarse grid, while $\Pi_f^{\perp}$ extracts the complementary unresolved component.

Theorem 1 (Orthogonal spectral decomposition) *Let R_f and P_f be the spectral restriction and prolongation operators in Eq. 2, and let Π_f and $\Pi_f^{\perp}$ denote the resolved- and unresolved-space projectors defined in Eqs. 3 and 4. For every fine-grid state u, define*

$$v = R_f(u), \qquad u_{\mathrm{lo}} = \Pi_f(u) = P_f(v), \qquad u_{\mathrm{hi}} = \Pi_f^{\perp}(u). \tag{5}$$

Then

$$u = u_{\mathrm{lo}} + u_{\mathrm{hi}}, \qquad \langle u_{\mathrm{lo}}, u_{\mathrm{hi}} \rangle_2 = 0, \qquad R_f(u_{\mathrm{hi}}) = 0. \tag{6}$$

Thus, the two state components occupy complementary orthogonal spectral subspaces. The proof is deferred to Appendix A.

The decomposition separates the state into orthogonal resolved and unresolved components, but their temporal dynamics remain coupled through the underlying PDE. Here, R_f maps the state to the coarse grid for numerical evolution, while Π_f and $\Pi_f^{\perp}$ identify the resolved and unresolved components within the fine-grid state space.

3.3 Hybrid Evolution

Design principle. Before specifying the discrete updates, we derive the continuous-time dynamics that motivate the two branches. Let $\mathcal{N}(u)$ denote the right-hand side of the fine-grid PDE,

$$\frac{\mathrm{d}u}{\mathrm{d}t} = \mathcal{N}(u). \tag{7}$$

Using the orthogonal decomposition in Eq. 5, and because the projectors are fixed in time, projecting Eq. 7 onto the two complementary subspaces gives the following coupled dynamics:

$$\frac{\mathrm{d}u_{\mathrm{lo}}}{\mathrm{d}t} = \underbrace{\Pi_f \mathcal{N}(u_{\mathrm{lo}})}_{\text{resolved dynamics}} + \underbrace{[\Pi_f \mathcal{N}(u_{\mathrm{lo}} + u_{\mathrm{hi}}) - \Pi_f \mathcal{N}(u_{\mathrm{lo}})]}_{\text{unresolved-to-resolved feedback}}, \tag{8}$$

$$\frac{\mathrm{d}u_{\mathrm{hi}}}{\mathrm{d}t} = \underbrace{\Pi_f^{\perp} \mathcal{N}(u_{\mathrm{lo}} + u_{\mathrm{hi}})}_{\text{unresolved dynamics}}, \tag{9}$$

These equations expose two distinct but coupled roles. The coarse-grid solver provides a numerical approximation to the resolved dynamics, while the closure module C_θ corrects the resulting resolved update, including both unresolved-scale feedback and discrepancies introduced by coarse numerical evolution. The refinement module H_ϕ models the temporal evolution of the unresolved component. Although their outputs are assigned to complementary subspaces, both updates can depend on information from the coupled resolved and unresolved state.

Resolved-state update. The coarse solver first advances the resolved state and approximates its intrinsic dynamics. The closure module C_θ then predicts the unresolved-to-resolved correction from the states before and after this solver step together with the current unresolved state. The correction is restricted to the resolved subspace by R_f:

$$\widetilde{v}^{n+1} = \mathcal{S}_{\Delta t}(v^n), \qquad v^{n+1} = \widetilde{v}^{n+1} + R_f\big(C_\theta(P_f(v^n), P_f(\widetilde{v}^{n+1}), u^n_{\mathrm{hi}}; \Delta t)\big)\,. \tag{10}$$

Because prolongation of this correction lies in the range of $\Pi_f = P_f \circ R_f$, the closure update modifies only the resolved subspace. The unresolved state u^n_{hi} provides the fine-scale context needed to model cross-scale feedback.

Unresolved-state update and reconstruction. The refinement module H_ϕ predicts a candidate increment for the unresolved state using the post-solver resolved state and the current unresolved component. We project this increment onto the unresolved subspace and scale it by a learnable gate α,

$$u^{n+1}_{\mathrm{hi}} = u^n_{\mathrm{hi}} + \alpha \Pi^{\perp}_f\big(H_\phi(P_f(\widetilde{v}^{n+1}), u^n_{\mathrm{hi}}; \Delta t)\big)\,, \tag{11}$$

$$\widehat{u}^{n+1} = P_f(v^{n+1}) + u^{n+1}_{\mathrm{hi}}. \tag{12}$$

The prolonged resolved state and the updated unresolved state are then recombined to form the next fine-resolution prediction. Repeating this coupled update produces the autoregressive trajectory. The complementary output constraints prevent refinement from modifying resolved modes while allowing both branches to exploit cross-scale information.

Neural architecture. We parameterize C_θ and H_ϕ with fully convolutional residual networks using multi-scale receptive fields. Both branches employ dilated convolutions, with different dilation configurations. Detailed architectures are provided in Appendix B.4.

4 Experiments

4.1 Setup

Datasets. We evaluate HybridOSS on three PDE benchmarks of increasing spatial dimensionality and computational complexity: the 1D Kuramoto–Sivashinsky (KS) equation, $\partial_t u + u\partial_x u + \partial_{xx} u + \partial_{xxxx} u = 0$, discretized on 512 grid points; the 2D Allen–Cahn (AC) equation, $\partial_t u = \epsilon^2 \nabla^2 u + u - u^3$, on a 128×128 grid; and the 3D incompressible Navier–Stokes (NS) equations for decaying homogeneous isotropic turbulence on a 64^3 grid. For KS and AC, we use existing solver implementations built on the PhiFlow framework (Holl & Thuerey, 2024; Mezidi et al., 2026), while the NS dataset is generated using our differentiable PIMPLE solver implemented within a differentiable OpenFOAM Towara (2018); Gonzalez-Sieiro et al. (2025). Detailed simulation parameters, data generation procedures, and dataset splits are provided in Appendix B.

Baselines. We comprehensively compare HybridOSS with 9 baselines, including solver-only, neural surrogate models: FNO (Li et al., 2021a), ResNet (He et al., 2016), Transolver (Wu et al., 2024), differentiable hybrid models: SITL(Um et al., 2020), INC(Wei et al., 2025), HybridDR(Mezidi et al., 2026),HybFB-pre(Mezidi et al., 2026) and HybFB-post(Mezidi et al., 2026). For the specific implementation of differentiable hybrid models, we follow the approach of (Mezidi et al., 2026).

Metric. We evaluate predictions using the global, resolved, and unresolved errors. For a predicted state $\widehat{u}$ and reference state u, the global relative L_2 error is

$$E_{\mathrm{global}} = \frac{\|\widehat{u} - u\|_2}{\|u\|_2}. \tag{13}$$

Following the decomposition in Eq. 5, the resolved and unresolved relative L_2 errors are then defined as

$$E_{\mathrm{resolved}} = \frac{\|\widehat{u}_{\mathrm{lo}} - u_{\mathrm{lo}}\|_2}{\|u_{\mathrm{lo}}\|_2}, \qquad E_{\mathrm{unresolved}} = \frac{\|\widehat{u}_{\mathrm{hi}} - u_{\mathrm{hi}}\|_2}{\|u_{\mathrm{hi}}\|_2}. \tag{14}$$

The resolved error indicates whether the model correctly recovers the global structure and large-scale organization of the physical field. Meanwhile, the unresolved error reflects whether the model recovers small-scale, high-gradient, and rapidly varying structures.

Table 1: Prediction errors of different methods on the KS and AC datasets. Lower is better for global, resolved, and unresolved errors. Best results are shown in bold, and second-best results are underlined. "–" denotes unavailable results because the method requires modifications to the numerical solver for which source code is unavailable; "NA" indicates a global error close to 1.

	KS			AC		
Model	Global ↓	Resolved ↓	Unresolved ↓	Global ↓	Resolved ↓	Unresolved ↓
Solver-only	0.2541	0.0637	**0.2603**	0.4836	0.2323	58.1416
FNO	NA	NA	NA	0.7194	0.4525	1105.9
ResNet	0.3188	0.0992	89.2594	NA	NA	NA
Transolver	0.4721	0.2245	11.7532	0.8526	0.7270	3.9481
SITL	<u>0.1241</u>	<u>0.0155</u>	0.7173	<u>0.0566</u>	0.0018	0.5358
INC	0.1305	0.0171	0.7320	–	–	–
HybFB-pre	0.1748	0.0309	0.5527	0.0712	0.0023	<u>0.3437</u>
HybFB-post	0.2176	0.0478	3.9642	0.0813	0.0045	1.6668
HybridDR	0.1756	0.0312	0.5708	0.0755	<u>0.0013</u>	10.1463
HybridOSS (ours)	**0.0858**	**0.0097**	<u>0.4756</u>	**0.0173**	**0.0004**	**0.1709**

Implementations. To ensure fair comparisons, we keep the numbers of trainable parameters within the same order of magnitude across methods. The competing hybrid methods operate at their native solver resolution and do not introduce a coarsening factor. For HybridOSS, the coarsening factor f controls the resolution of the solver relative to the fine-resolution state and is fixed per dataset, with $f = 16$ for KS, $f = 4$ for AC, and $f = 8$ for NS. For the evaluation of resolved and unresolved errors, all methods use the same spectral decomposition as HybridOSS with the dataset-specific coarsening factor used to train HybridOSS. All experiments are conducted on a single NVIDIA 4090 GPU and repeated three times with results averaged. Detailed experimental settings and hyperparameter configurations are provided in Appendix B.4.

4.2 Main Results

Accuracy. Table 1 shows that HybridOSS achieves the lowest reported global error on both KS and AC, reaching 0.0858 and 0.0173, respectively. Compared with the strongest hybrid baseline, SITL, these values correspond to relative reductions of approximately 31% on KS and 69% on AC. The hybrid models also substantially outperform the neural-only baselines in the autoregressive setting, indicating the benefit of retaining the numerical solver as a physical backbone for long-horizon rollouts.

The improvement is consistent across the resolved and unresolved subspaces. HybridOSS obtains the lowest resolved error on both datasets, 0.0097 on KS and 0.0004 on AC, and also achieves the best unresolved error among the hybrid methods, 0.4756 and 0.1709, respectively. These results indicate that the proposed architecture accurately represents both large-scale resolved dynamics and small-scale unresolved details. Consistent with these quantitative results, Fig. 2(a) and (d) shows that HybridOSS more faithfully preserves both resolved structures and unresolved details, indicating that the improvement reflects better spatial fidelity rather than only lower aggregate error.

Cost. Fig. 2 (b) and (c) show a clear scale-dependent trade-off. On AC with $f = 4$, HybridOSS reduces peak GPU memory by 20.0% and training time by 36.3% compared with the most efficient competing hybrid methods. On the smaller 1D KS problem, HybridOSS still reduces memory, but incurs higher training time. This suggests that, for small problems, the additional projection overhead can outweigh the savings from coarse-grid evolution, whereas the computational benefits of coarsening become more pronounced as problem size increases. We further examine this accuracy–efficiency trade-off by varying the coarsening factor in Section 4.3.

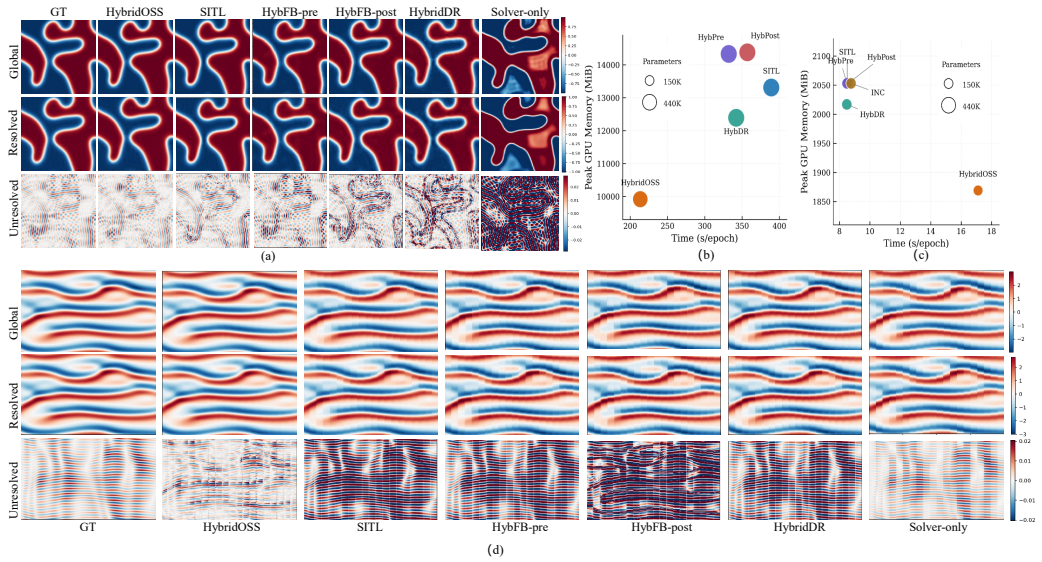


Figure 2: Comparison of prediction accuracy and computational cost across hybrid models on the AC and KS datasets. (a) Global, resolved-state, and unresolved-state prediction on AC. (b) Computational cost of different hybrid models on AC. (c) Computational cost of different hybrid models on KS. (d) Global, resolved-state, and unresolved-state prediction on KS.

Table 2: Component ablations and full-state correction comparison on KS and AC. "w/o" indicates that the corresponding component is removed. Lower values indicate better performance for global, resolved, and unresolved errors. Best results are shown in bold, and the second are underlined.

	KS			AC		
Variant	Global ↓	Resolved ↓	Unresolved ↓	Global ↓	Resolved ↓	Unresolved ↓
w/o both modules	0.3859	0.1762	0.8350	0.5369	0.2898	2.0467
w/o closure	0.3830	0.1762	1.4859	0.5368	0.2898	0.9725
w/o refinement	0.1163	<u>0.0090</u>	0.8350	<u>0.0182</u>	**0.0004**	2.0470
w/o $\Pi_f^{\perp}$	<u>0.0882</u>	**0.0067**	0.9968	0.0190	<u>0.0007</u>	0.4983
Full-state	0.0975	0.0105	<u>0.5616</u>	0.0252	0.0012	<u>0.4246</u>
HybridOSS (full)	**0.0858**	0.0097	**0.4756**	**0.0173**	**0.0004**	**0.1709**

4.3 Ablation Study

While the comparisons in Section 4.2 evaluate HybridOSS as a complete method against existing approaches, the following controlled ablations isolate the contributions of its individual design components.

Effect of closure and refinement modules. As shown in Table 2, the ablation results demonstrate that the closure and refinement modules play complementary roles. Removing the closure module substantially increases both global and resolved errors on both PDEs, while adding refinement without closure does not recover the resolved dynamics. In contrast, removing refinement mainly degrades the unresolved error, with only a minor effect on the resolved component. These results indicate that the closure module primarily corrects the resolved dynamics affected by unresolved scales, whereas the refinement module is responsible for explicitly evolving the unresolved state.

Effect of the unresolved-space projection $\Pi_f^{\perp}$. Comparing the "w/o $\Pi_f^{\perp}$" row with "HybridOSS (full)" in Table 2 shows that removing the unresolved-space projection substantially increases the unresolved error on both PDEs. On KS, the resolved error even decreases slightly while the global

Table 3: Accuracy–efficiency trade-off under different coarsening factors f on KS and AC. Entries report prediction error and runtime per training epoch. Lower values are better. Best results are shown in bold, and second-best results are underlined. "Not converged" indicates that training became unstable and terminated before completion; "–" indicates that runtime was not measured for the corresponding setting.

f	KS Global Error	KS time (s/epoch)	AC Global Error	AC time (s/epoch)
1	**0.0047**	17.51	<u>0.0158</u>	348.50
2	**0.0047**	17.37	**0.0151**	263.83
4	<u>0.0048</u>	17.29	0.0173	<u>213.53</u>
8	0.0057	17.24	0.0288	**192.35**
16	0.0858	<u>17.11</u>	Not converged	–
32	0.1752	**17.01**	Not converged	–

error changes little, indicating that an unrestricted refinement update can improve the aggregate or resolved prediction by leaking into the resolved subspace, at the expense of unresolved accuracy. This also shows that a low global error does not necessarily mean that both the resolved and unresolved components are predicted correctly. The unresolved-space constraint therefore acts as an explicit mechanism for preventing such interference between the two branches.

Comparison with full-state correction. The full-state correction uses a single capacity-matched network to predict an unrestricted fine-grid correction that is added directly to $P_f(v^{n+1})+u^n_{\mathrm{hi}}$. Since the correction is not projected onto either subspace and is supervised against the full fine-resolution target, it can jointly correct the resolved component and reconstruct the unresolved component. As shown in Table 2, full-state correction is consistently less accurate than HybridOSS. This suggests that the advantage of HybridOSS explicitly assigning the resolved and unresolved corrections to complementary subspaces provides a useful architectural inductive bias.

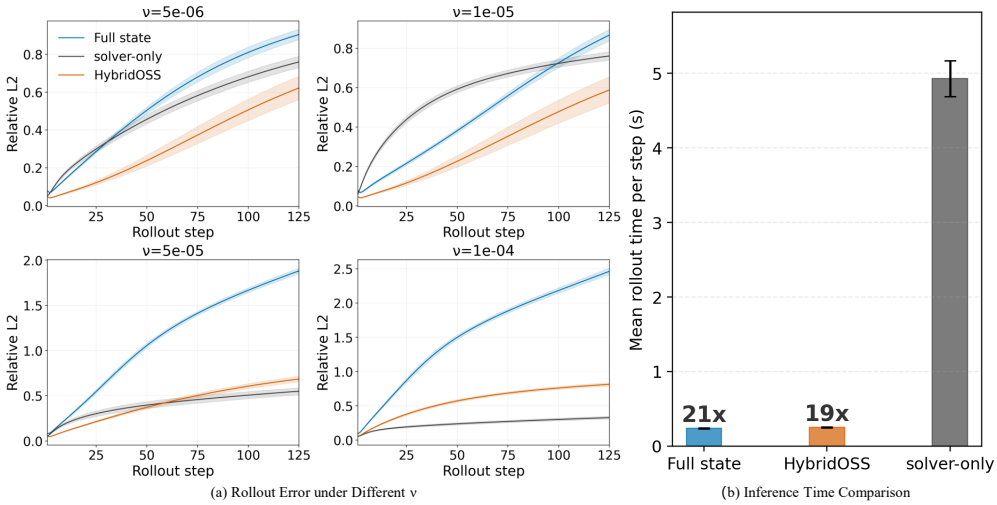
Effect of coarsening factor f. As shown in Table 3, the coarsening factor controls the balance between numerical and learned evolution. Smaller f gives the solver a finer grid, retains more scale information, and leaves a smaller unresolved space for the refinement network to model. Conversely, larger f shifts more dynamics to the learned branch and can cause sharp error growth or solver failure once important structures are no longer resolved. Reducing f also increases the computational cost of the coarse solver, making f a problem- and budget-dependent accuracy–efficiency trade-off rather than a universal optimum. HybridOSS makes this trade-off explicit by retaining the unresolved state and assigning its evolution to a dedicated refinement module.

Overall, the ablations support the proposed division of roles in which closure corrects the resolved dynamics, refinement evolves the unresolved state, and the unresolved-space projection $\Pi_f^{\perp}$ preserves their separation, while the coarsening factor controls the balance between prediction accuracy and solver efficiency.

4.4 Evaluation on Differentiable OpenFOAM

We further evaluate HybridOSS on the 3D incompressible Navier–Stokes (NS) equations for decaying homogeneous isotropic turbulence using differentiable OpenFOAM (Towara, 2018; Gonzalez-Sieiro et al., 2025). The model is trained only at viscosity $\nu = 10^{-5}$ with a three-step rollout and evaluated over 125 autoregressive steps, more than $40\times$ the training rollout horizon. Generalization is further evaluated at several unseen viscosity values without retraining, with detailed configurations provided in the Appendix B.3.

Long-horizon stability. As shown in Fig. 3(a), at the training viscosity $\nu = 10^{-5}$, HybridOSS consistently maintains lower error throughout the 125-step rollout. In contrast, the error of full-state correction accumulates progressively and exceeds that of solver-only at later steps. By the end of the rollout, HybridOSS reduces the prediction error by about 23% relative to solver-only and 32% relative to full-state correction. The advantage persisting far beyond the three-step training horizon



(a) Rollout Error under Different ν

(b) Inference Time Comparison

Figure 3: Prediction accuracy and inference efficiency across different ν values. (a) Evolution of prediction error over rollout time for HybridOSS, full-state correction, and solver-only on test sets with different ν values. (b) Per-step inference time of the three methods, with annotations showing the corresponding speedup relative to solver-only.

indicates that separating resolved and unresolved corrections reduces long-term error accumulation during autoregressive prediction.

Viscosity generalization. Fig. 3(a) also shows that HybridOSS is more robust to viscosity shifts than full-state correction. At the unseen viscosity $\nu = 5 \times 10^{-6}$, it remains better than both solver-only and full-state correction throughout the entire rollout, reducing the final error by about 18% relative to solver-only. As the viscosity moves farther from the training condition, the advantage over solver-only gradually decreases. At $\nu = 5 \times 10^{-5}$, HybridOSS remains more accurate during the earlier part of the rollout but is eventually overtaken by solver-only, while at $\nu = 10^{-4}$ the solver-only baseline becomes preferable. Nevertheless, HybridOSS remains consistently more accurate than full-state correction across these unseen conditions.

Accuracy–efficiency trade-off. As shown in Fig. 3(b), HybridOSS achieves approximately a $19\times$ inference speedup over solver-only. This is particularly important for solver-in-the-loop hybrid models such as SITL, where repeated differentiable fine-grid solver evaluations make training increasingly expensive on large three-dimensional problems and limit practical scalability. HybridOSS alleviates this bottleneck by evolving only the resolved state numerically on a coarse grid, while retaining fine-resolution dynamics through learned refinement. Despite some additional neural cost over full-state correction, it provides a more favorable accuracy–efficiency trade-off through improved long-horizon accuracy and substantially reduced dependence on fine-grid solver evaluations.

Overall, these results support the central motivation of HybridOSS. By combining coarse-grid numerical evolution with learned fine-scale dynamics, HybridOSS maintains accurate long-horizon predictions while substantially reducing inference cost, demonstrating its potential for efficient 3D PDE simulation.

5 Conclusion

We introduced HybridOSS, a differentiable hybrid PDE framework that decouples solver and state resolutions through orthogonal scale splitting. By combining coarse-grid numerical evolution with dedicated closure and refinement modules, HybridOSS improves multiscale prediction while reducing computational cost. Experiments on differentiable OpenFOAM further demonstrate stable long-horizon prediction, viscosity generalization, and substantial inference acceleration, supporting efficient fine-resolution PDE simulation without fine-grid numerical evolution.

AI USE STATEMENT

We used generative AI tools only to improve the language, grammar, and readability of the manuscript. AI tools were not used to generate the research ideas, develop the method, design the experiments, analyze the results, or draw the scientific conclusions. All AI-assisted edits were reviewed and revised by the authors, who take full responsibility for the final content of this work.

REFERENCES

Giancarlo Alfonsi. Reynolds-averaged navier–stokes equations for turbulence modeling. 2009.

Benedikt Alkin, Andreas Fürst, Simon Schmid, Lukas Gruber, Markus Holzleitner, and Johannes Brandstetter. Universal physics transformers: a framework for efficiently scaling neural operators. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, pp. 25152–25194, 2024.

Yuri Bazilevs, Victor M Calo, John A Cottrell, Thomas JR Hughes, Alessandro Reali, and G23614751169 Scovazzi. Variational multiscale residual-based turbulence modeling for large eddy simulation of incompressible flows. *Computer methods in applied mechanics and engineering*, 197(1-4):173–201, 2007.

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

Christophe Devulder and Martine Marion. A class of numerical algorithms for large time integration: the nonlinear galerkin methods. *SIAM Journal on numerical analysis*, 29(2):462–483, 1992.

Jesus Gonzalez-Sieiro, David Pardo, Vincenzo Nava, Victor M Calo, and Markus Towara. Reducing spatial discretization error on coarse cfd simulations using an openfoam-embedded deep learning framework. *Engineering with Computers*, 41(3):1699–1720, 2025.

Ayoub Gouasmi, Eric J Parish, and Karthik Duraisamy. A priori estimation of memory effects in reduced-order models of nonlinear systems using the mori–zwanzig formalism. *Proceedings. Mathematical, Physical, and Engineering Sciences*, 473(2205):20170385, 2017.

Zhongkai Hao, Zhengyi Wang, Hang Su, Chengyang Ying, Yinpeng Dong, Songming Liu, Ze Cheng, Jian Song, and Jun Zhu. Gnot: A general neural operator transformer for operator learning. In *International conference on machine learning*, pp. 12556–12569. PMLR, 2023.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Philipp Holl and Nils Thuerey. ϕ_{flow}: Differentiable simulations for machine learning. In *ICML Workshop on Differentiable Almost Everything*, 2024.

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

Thomas JR Hughes, Gonzalo R Feijóo, Luca Mazzei, and Jean-Baptiste Quincy. The variational multiscale method—a paradigm for computational mechanics. *Computer methods in applied mechanics and engineering*, 166(1-2):3–24, 1998.

Ruoxi Jiang, Peter Y Lu, Elena Orlova, and Rebecca Willett. Training neural operators to preserve invariant measures of chaotic attractors. *Advances in Neural Information Processing Systems*, 36: 27645–27669, 2023.

Dmitrii Kochkov, Jamie A Smith, Ayya Alieva, Qing Wang, Michael P Brenner, and Stephan Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21):e2101784118, 2021.

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

Remi Lam, Alvaro Sanchez-Gonzalez, Matthew Willson, Peter Wirnsberger, Meire Fortunato, Ferran Alet, Suman Ravuri, Timo Ewalds, Zach Eaton-Rosen, Weihua Hu, et al. Learning skillful medium-range global weather forecasting. *Science*, 382(6677):1416–1421, 2023.

Myoungkyu Lee and Robert D Moser. Direct numerical simulation of turbulent channel flow up to. *Journal of fluid mechanics*, 774:395–415, 2015.

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

Zijie Li, Dule Shu, and Amir Barati Farimani. Scalable transformer for pde surrogate modeling. *Advances in Neural Information Processing Systems*, 36:28010–28039, 2023.

Zongyi Li, Nikola B. Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew M. Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations. *CoRR*, abs/2003.03485, 2020. URL `https://arxiv.org/abs/2003.03485`.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021a.

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*, 2021b.

Benjamin 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.

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

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

Huakun Luo, Haixu Wu, Hang Zhou, Lanxiang Xing, Yichen Di, Jianmin Wang, and Mingsheng Long. Transolver++: An accurate neural solver for pdes on million-scale geometries. *arXiv preprint arXiv:2502.02414*, 2025.

Abdel-Rahim Mezidi, Jordan Patracone, and Amaury Habrard. Proximal splitting methods for hybrid differentiable models. In *Forty-Third International Conference on Machine Learning*, 2026.

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.

Parviz Moin and Krishnan Mahesh. Direct numerical simulation: a tool in turbulence research. *Annual review of fluid mechanics*, 30(1):539–578, 1998.

Chris Pedersen, Laure Zanna, and Joan Bruna. Thermalizer: Stable autoregressive neural emulation of spatiotemporal chaos. *arXiv preprint arXiv:2503.18731*, 2025.

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. FiLM: Visual reasoning with a general conditioning layer. In *AAAI Conference on Artificial Intelligence*, 2018.

Ugo Piomelli. Large-eddy simulation: achievements and challenges. *Progress in aerospace sciences*, 35(4):335–362, 1999.

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International conference on machine learning*, pp. 8459–8468. Pmlr, 2020.

Claude E Shannon. Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21, 1949.

Kimberly L Stachenfeld, Drummond Buschman Fielding, Dmitrii Kochkov, Miles D Cranmer, Tobias Pfaff, Jonathan Godwin, Can Cui, Shirley Ho, Peter W Battaglia, and Alvaro Sanchez-Gonzalez. Learned simulators for turbulence. In *ICLR*, 2022.

Markus Towara. *Discrete adjoint optimization with OpenFOAM*. PhD thesis, Dissertation, RWTH Aachen University, 2018, 2018.

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, pp. 6111–6122, 2020.

Tian Wang and Chuang Wang. Latent neural operator for solving forward and inverse pde problems. *Advances in Neural Information Processing Systems*, 37:33085–33107, 2024.

Hao Wei, Anna Franz, Benjamin M List, and Nils Thuerey. INC: An indirect neural corrector for auto-regressive hybrid PDE solvers. In *Advances in Neural Information Processing Systems*, 2025.

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

Hang Zhou, Haixu Wu, Haonan Shangguan, Yuezhou Ma, Huikun Weng, Jianmin Wang, and Mingsheng Long. Transolver-3: Scaling up transformer solvers to industrial-scale geometries. *arXiv preprint arXiv:2602.04940*, 2026.

A Proof of the Orthogonal Spectral Decomposition

Proof. For $f > 1$, let M_f denote the diagonal binary mask that selects the Fourier modes retained by the spectral restriction operator. Restriction followed by prolongation is therefore equivalent to masking the fine-grid Fourier coefficients and transforming them back to the physical domain:

$$\Pi_f = P_f R_f = \mathcal{F}_N^{-1} M_f \mathcal{F}_N. \tag{15}$$

Since M_f is a binary diagonal mask,

$$M_f^2 = M_f, \qquad M_f^* = M_f. \tag{16}$$

Because the discrete Fourier transform is unitary up to its normalization convention, we obtain

$$\Pi_f^2 = \Pi_f, \qquad \Pi_f^* = \Pi_f. \tag{17}$$

Hence, Π_f is the orthogonal projector onto the retained spectral subspace, while

$$\Pi_f^\perp = I - \Pi_f \tag{18}$$

is the orthogonal projector onto its complementary subspace.

For any fine-grid state u, define

$$u_{\mathrm{lo}} = \Pi_f u, \qquad u_{\mathrm{hi}} = \Pi_f^\perp u. \tag{19}$$

Then

$$u = \Pi_f u + (I - \Pi_f)u = u_{\mathrm{lo}} + u_{\mathrm{hi}}. \tag{20}$$

Moreover,

$$\langle u_{\mathrm{lo}}, u_{\mathrm{hi}} \rangle_2 = \langle \Pi_f u, (I - \Pi_f) u \rangle_2 \tag{21}$$
$$= \langle u, \Pi_f (I - \Pi_f) u \rangle_2 \tag{22}$$
$$= 0, \tag{23}$$

because

$$\Pi_f (I - \Pi_f) = \Pi_f - \Pi_f^2 = 0. \tag{24}$$

Finally, u_{hi} contains only Fourier modes excluded by M_f. Since R_f retains only the modes selected by M_f, these unresolved modes are removed under restriction, giving

$$R_f(u_{\mathrm{hi}}) = 0. \tag{25}$$

Therefore,

$$u = u_{\mathrm{lo}} + u_{\mathrm{hi}}, \qquad \langle u_{\mathrm{lo}}, u_{\mathrm{hi}} \rangle_2 = 0, \qquad R_f(u_{\mathrm{hi}}) = 0, \tag{26}$$

which establishes Eq. 6.

For $f = 1$, no Fourier modes are discarded, so $\Pi_f = I$ and $\Pi_f^{\perp} = 0$, and the result follows immediately. □

B Implementation Details

In this section, we provide detailed descriptions of our experimental setup, including the solver definitions, dataset generation, and implementation.

Table 4: Summary of dataset construction and training configurations for KS, AC, and NS, including spatial resolution, trajectory split, training samples, solver time steps, and autoregressive rollout lengths. HF and LF denote the high- and low-fidelity solvers, respectively.

Dataset	Grid resolution	Trajectory split	Training samples	Time step (HF/LF)	Rollout (Train/Test)
KS	512	9/3/6	189	0.01/0.02	50/50
AC	128^2	700/150/150	700	0.02/1.5	10/10
NS	64^3	1/0/40	123	0.001/0.008	3/125

B.1 Allen–Cahn

Solver Definition. The Allen–Cahn (AC) equation is a prototypical phase-field model describing the evolution of an order parameter $u(\mathbf{x}, t)$ in a bistable system:

$$\partial_t u = \epsilon^2 \nabla^2 u - (u^3 - u), \tag{27}$$

where $\epsilon > 0$ controls the interface width, and the nonlinear term $-(u^3 - u)$ corresponds to a double-well potential favoring states near $u \approx \pm 1$.

We follow the PhiFlow-based differentiable solver implementation used in prior work (Holl & Thuerey, 2024; Mezidi et al., 2026), with periodic boundary conditions. The solver decomposes the Allen–Cahn dynamics into diffusion and reaction steps. For the diffusion step, the Laplacian term can be advanced using either an explicit or implicit Euler scheme. The implicit scheme uses PhiFlow's `diffuse.implicit` operator to solve the corresponding linear system, whereas the explicit scheme uses `diffuse.explicit` for explicit diffusion. For the reaction step, the cubic nonlinear term is treated explicitly using forward Euler:

$$u^{n+1} = u^n + \Delta t \left(-(u^n)^3 + u^n \right). \tag{28}$$

The resulting solver is fully differentiable and therefore allows gradients to propagate through the numerical evolution during model training.

Dataset generation. We generate the AC dataset on a 128×128 periodic domain with $\epsilon = 0.02$ and domain size $L = 1$. The numerical solver uses an internal time step of $\Delta t = 0.02$ with implicit diffusion. States are recorded every 0.1 time units over a total simulation time of 25, corresponding to five solver substeps between two consecutive saved states and 251 snapshots per trajectory, including the initial state. We generate 1000 trajectories from independently sampled initial conditions. To obtain spatially smooth initial fields, Gaussian random noise is first low-pass filtered in Fourier space using a Gaussian spectral filter. Each field is then centered and normalized to unit variance, followed by a pointwise transformation

$$u^0 = \tanh(0.75\,\widetilde{u}), \tag{29}$$

where $\widetilde{u}$ denotes the normalized smooth random field. We use a spectral cutoff parameter of 0.08 and a fixed random seed of 42 for reproducibility. The resulting dataset is stored in single precision with shape

$$\mathcal{D}_{\mathrm{AC}} \in \mathbb{R}^{1000\times 251\times 128\times 128}, \tag{30}$$

where the four dimensions correspond to trajectory, time, and the two spatial dimensions, respectively.

Training Details. Following the experimental setup of Mezidi et al. (2026), we use 700 trajectories for training, 150 for validation, and 150 for testing. For each trajectory, the 10th frame is used as the initial state and the 160th frame as the target state.

The high-fidelity reference trajectories are generated using the Allen–Cahn solver with a time step of $\Delta t = 0.02$. In contrast, the low-fidelity physical solver operates with a larger time step of $\Delta t = 1.5$. Starting from the sampled initial state, the model performs 10 autoregressive rollout steps to reach the corresponding prediction horizon.

B.2 Kuramoto–Sivashinsky

Solver Definition. The Kuramoto–Sivashinsky (KS) equation is a fourth-order nonlinear PDE exhibiting spatiotemporal chaos:

$$\partial_t u + u\partial_x u + \partial_{xx} u + \partial_{xxxx} u = 0, \tag{31}$$

where the nonlinear advection term $u\partial_x u$ couples with the second- and fourth-order derivatives to produce complex dynamics. Our implementation follows the differentiable pseudo-spectral KS solver used in prior work (Wei et al., 2025; Mezidi et al., 2026). Spatial derivatives are computed in Fourier space using PyTorch FFT operations. For a periodic domain of length L, the wavenumbers are

$$k = \frac{2\pi n}{L}, \qquad n \in \left\{-\frac{N}{2}, \ldots, \frac{N}{2} - 1\right\}, \tag{32}$$

where N is the spatial resolution. The linear operator in Fourier space is

$$\widehat{L}_k = -(ik)^2 - (ik)^4, \tag{33}$$

the nonlinear advection term is evaluated in physical space as $-\frac{1}{2}\partial_x(u^2)$ and transformed back to Fourier space.

We use different exponential time-integration schemes for high- and low-fidelity evolution. The low-fidelity solver used by HybridOSS employs first-order exponential time differencing (ETD1). Defining

$$E_k = e^{\widehat{L}_k \Delta t}, \qquad \phi_{1,k} = \begin{cases} \dfrac{E_k - 1}{\widehat{L}_k}, & \widehat{L}_k \neq 0, \\ \Delta t, & \widehat{L}_k = 0, \end{cases} \tag{34}$$

the ETD1 update is

$$\widehat{u}_k^{n+1} = E_k \widehat{u}_k^n + \phi_{1,k} \widehat{\mathcal{N}(u^n)}_k, \tag{35}$$

where $\mathcal{N}(u) = -u\partial_x u$.

The high-fidelity reference trajectories are generated using the second-order exponential time-differencing Runge–Kutta scheme (ETRK2). It first computes an ETD1 predictor,

$$\widehat{u}_k^* = E_k \widehat{u}_k^n + \phi_{1,k} \widehat{\mathcal{N}(u^n)}_k, \tag{36}$$

and then reevaluates the nonlinear term at the predicted state. The corrected solution is

$$\widehat{u}_k^{n+1} = \widehat{u}_k^* + \phi_{2,k}\left(\widehat{\mathcal{N}(u^*)}_k - \widehat{\mathcal{N}(u^n)}_k\right), \tag{37}$$

where

$$\phi_{2,k} = \begin{cases} \dfrac{E_k - 1 - \widehat{L}_k \Delta t}{\widehat{L}_k^2 \Delta t}, & \widehat{L}_k \neq 0, \\ \dfrac{\Delta t}{2}, & \widehat{L}_k = 0. \end{cases} \tag{38}$$

Thus, ETRK2 is used to generate the high-fidelity reference data, whereas ETD1 provides the lower-fidelity numerical solver used in HybridOSS.

Dataset generation. The KS trajectories are generated on a one-dimensional grid with a spatial resolution of $N = 512$. Each trajectory is initialized as

$$u(x, 0) = \cos(2\pi x) + \alpha \sin(2\pi x) + \eta(x), \tag{39}$$

where α is sampled from $\mathcal{U}(-0.5, 0.5)$ and $\eta(x)$ is a normalized Gaussian perturbation. Before recording the trajectory, the initial state is evolved for 2000 warm-up steps to reduce dependence on the prescribed initial-state form.

After warm-up, the high-fidelity trajectory is generated using the ETRK2 solver with a time step of $\Delta t = 0.01$ over a simulation horizon of $T = 100$. Since states are recorded every 0.01 time units, each trajectory contains $10{,}000$ time-advancement steps and $10{,}001$ snapshots, including the state immediately after warm-up as the initial recorded state. Each trajectory therefore has the form

$$\mathcal{U}_{\mathrm{KS}} = \{u^0, u^1, \ldots, u^{10000}\}, \qquad u^n \in \mathbb{R}^{512}. \tag{40}$$

We additionally vary the physical domain length L. For training and validation,

$$\frac{L}{2\pi} \in \{6.4,\ 7.2,\ 8.0,\ 8.8,\ 9.6,\ 10.4\}, \tag{41}$$

with two independently initialized trajectories for each domain size. The test set uses

$$\frac{L}{2\pi} \in \{6.0,\ 8.4,\ 10.8\}, \tag{42}$$

with two trajectories for each domain size, resulting in 6 test trajectories.

The trajectories are generated using the differentiable KS solver with the `ETRK2` time-integration scheme and double stepping. Each split is stored together with its corresponding domain lengths and simulation metadata. For a spatial resolution N_x, the resulting trajectory tensor has the form

$$\mathcal{U}_{\mathrm{KS}} \in \mathbb{R}^{N_{\mathrm{traj}} \times N_t \times N_x}. \tag{43}$$

Training Details. Following the experimental setup of Mezidi et al. (2026), we use 9 trajectories for training, 3 trajectories for validation, and 6 trajectories for testing. For each trajectory, we sample states from the 1000th to the 3000th frame at an interval of 100 frames, resulting in 21 samples per trajectory. This yields 189 training samples, 63 validation samples, and 126 test samples, for a total of 378 samples. The high-fidelity reference trajectories are generated using the ETRK2 solver with a time step of $\Delta t = 0.01$. In contrast, the low-fidelity physical solver uses ETD1 with a larger time step of $\Delta t = 0.02$. Starting from each sampled state, the model is evaluated through 50 autoregressive rollout steps to obtain the final prediction.

B.3 Incompressible Navier–Stokes

Solver Definition. We consider the incompressible Navier–Stokes equations

$$\partial_t \mathbf{u} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla p + \nu \nabla^2 \mathbf{u}, \tag{44}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{45}$$

where $\mathbf{u}$ denotes the velocity field, p is the pressure, and ν is the kinematic viscosity.

We solve the equations using a differentiable PIMPLE algorithm implemented within differentiable OpenFOAM (Towara, 2018; Gonzalez-Sieiro et al., 2025). PIMPLE combines pressure–velocity coupling with iterative predictor–corrector updates. At each time step, a provisional velocity field $\mathbf{u}^*$ is first obtained from the discretized momentum equation using the current pressure estimate:

$$\frac{\mathbf{u}^* - \mathbf{u}^n}{\Delta t} + (\mathbf{u}^n \cdot \nabla)\mathbf{u}^n = -\nabla p^n + \nu \nabla^2 \mathbf{u}^*. \tag{46}$$

The pressure is then updated by solving a pressure-correction equation derived from the incompressibility constraint,

$$\nabla \cdot \mathbf{u}^{n+1} = 0, \tag{47}$$

and the velocity field is corrected accordingly:

$$\mathbf{u}^{n+1} = \mathbf{u}^* - \Delta t \, \nabla p', \tag{48}$$

where p' denotes the pressure correction. The pressure and velocity corrections are iterated within each time step until the prescribed PIMPLE iterations are completed. The resulting solver preserves the pressure–velocity coupling of OpenFOAM while remaining differentiable, allowing gradients to propagate through the numerical evolution during training.

Dataset generation. The dataset is generated using OpenFOAM v2112 based on the standard `decayIsoTurb` tutorial case for three-dimensional decaying homogeneous isotropic turbulence. The simulations are performed in a periodic cubic domain of size

$$(0.09 \times 2\pi)^3 \approx (0.5655\,\mathrm{m})^3$$

on a $64 \times 64 \times 64$ grid. We use `pimpleFoam` for temporal evolution together with the Smagorinsky large-eddy simulation (LES) model for subgrid-scale turbulence. Four kinematic viscosities are considered,

$$\nu \in \{5 \times 10^{-6},\, 10^{-5},\, 5 \times 10^{-5},\, 10^{-4}\}.$$

The training trajectory is generated from the standard OpenFOAM `decayIsoTurb` case at $\nu = 10^{-5}$, using its original initial condition without the spectral perturbation described below. For evaluation, we additionally generate 10 trajectories for each of the four viscosity conditions, resulting in 40 test trajectories with randomized initial turbulent states.

To construct different initial conditions under the same viscosity while maintaining comparable initial energy levels, we perturb the tabulated energy spectrum $E(k)$ of the reference `decayIsoTurb` case. For each trajectory, each spectral coefficient is perturbed according to

$$\widetilde{E}(k_i) = E(k_i)(1 + \epsilon_i), \qquad \epsilon_i \in [-0.3, 0.3],$$

where the perturbations are deterministically generated from a trajectory-specific seed. The perturbed spectrum is subsequently renormalized as

$$E^{\mathrm{IC}}(k_i) = \widetilde{E}(k_i) \frac{\sum_j E(k_j)}{\sum_j \widetilde{E}(k_j)},$$

such that the total tabulated spectral energy is preserved while its distribution across wavenumbers varies among trajectories. The resulting spectrum is supplied to `createBoxTurb` to synthesize the corresponding isotropic turbulent velocity field.

For each realization, `createBoxTurb -createBlockMesh` first constructs the 64^3 periodic mesh, after which `createBoxTurb` generates the initial turbulent velocity field. Starting from this state, `pimpleFoam` advances the flow with a simulation time step of $\Delta t = 0.001\,\mathrm{s}$ for a total physical time of $1\,\mathrm{s}$. The flow fields are recorded every $0.008\,\mathrm{s}$, yielding 126 snapshots for each trajectory. The computational domain, spatial resolution, boundary conditions, turbulence model, and numerical schemes are kept fixed across all simulations, while only the viscosity and initial spectral realization are varied.

Training Details. Among the 41 generated trajectories, we use the standard OpenFOAM `decayIsoTurb` case at $\nu = 10^{-5}$ for training, while the remaining 40 trajectories are reserved for evaluation, including both unseen initial conditions and unseen viscosity settings. The high-fidelity OpenFOAM simulation is integrated with a time step of $\Delta t_{\mathrm{HF}} = 0.001\,\mathrm{s}$, whereas the low-fidelity solver used in the hybrid model advances with $\Delta t_{\mathrm{LF}} = 0.008\,\mathrm{s}$. Thus, each low-fidelity solver step corresponds to eight high-fidelity integration steps.

For training, the trajectory is divided into overlapping three-step windows with a stride of one, yielding 123 training samples. Each sample is used to optimize the model over a short local rollout, while adjacent samples share temporal states to maximize the use of the single training trajectory. At test time, no window-wise resetting is used. Starting from the initial state, the model is rolled out autoregressively for the full 125-step horizon, with each predicted state fed back as the input to the next step. This setting evaluates whether a model trained only on short-horizon transitions from a single trajectory can remain stable over long autoregressive rollouts and generalize to unseen turbulent realizations and viscosity conditions.

Table 5: Training and network configurations for the three datasets. $\mathcal{D}_{\mathrm{cl}}$ and $\mathcal{D}_{\mathrm{hi}}$ denote the dilation sets of the closure and high-frequency branches, respectively.

Configuration	KS	AC	NS
Training			
Epochs	200	200	200
Batch size	8	2	2
Initial learning rate	10^{-3}	10^{-3}	10^{-3}
Weight decay	10^{-5}	10^{-6}	10^{-5}
Gradient clipping	1.0	1.0	1.0
LR scheduler	Plateau	Plateau	Plateau
LR decay factor	0.75	0.5	0.5
LR patience	10	5	10
Minimum LR	10^{-5}	10^{-7}	10^{-6}
Intermediate loss	Yes	No	Yes
Loss	$\mathcal{L}_{MSE}$	$\mathcal{L}_{MSE}$	$\mathcal{L}_{RL_2}$
Hybrid architecture			
Coarsening factor f	16	4	8
Hidden channels	64	64	64
Closure channels	32	32	64
Closure blocks	1	1	3
$\mathcal{D}_{\mathrm{cl}}$	$\{1, 4, 16\}$	$\{1, 2, 4\}$	$\{1, 2, 4\}$
Refinement channels	64	64	64
Refinement blocks	1	1	3
$\mathcal{D}_{\mathrm{hi}}$	$\{1, 4, 8, 16, 32\}$	$\{1, 2, 4, 8\}$	$\{1, 2, 4\}$
Initial Refinement scale	0.1	0.1	0.1
Learnable scale	Yes	Yes	Yes
Δt conditioning	Yes	Yes	Yes

B.4 Implementations

Branch network architecture. We use two convolutional branches for the unresolved-state update and coarse-scale closure. In addition to their state inputs, both branches receive explicit spatial and physics-derived features. The refinement branch concatenates the post-solver resolved state and current unresolved state with the resolved-state gradient magnitude and Laplacian, the local unresolved-state energy, and a cross-scale interaction feature computed from the pointwise product of the resolved and unresolved states. The closure branch concatenates the pre- and post-solver resolved states and current unresolved state with the solver increment (the difference between the post- and pre-solver states), the post-solver gradient magnitude, and the local unresolved-state energy. These derived channels provide direct cues about sharp spatial variations, the update produced by the coarse solver, and local cross-scale activity. Both networks employ periodic 3×3 convolutions, GELU activations, residual multi-scale blocks, and dilated convolutions. The dilation sets $\mathcal{D}_{\mathrm{hi}}$ and $\mathcal{D}_{\mathrm{cl}}$, hidden widths, and number of residual blocks are configured separately for each dataset

to accommodate different spatial scales and problem dimensionalities. The solver time step is incorporated through FiLM (Perez et al., 2018) conditioning using $(\Delta t, \log \Delta t)$. For the NS cases, the viscosity ν is additionally encoded through FiLM conditioning to account for variations in the flow regime. The final prediction layers are zero-initialized so that training starts from the uncorrected numerical solver. The detailed network and training configurations for KS, AC, and NS are summarized in Table 5.

Implementation of other hybrid models. Following the implementation of Mezidi et al. (2026), we use a ResNet as the main backbone for the other hybrid baselines. The network width is fixed to 64 channels with 6 residual blocks across all experiments to ensure a consistent architecture and comparable model capacity.

Implementation of neural surrogates. For the purely neural surrogate baselines, the ResNet and FNO implementations follow Mezidi et al. (2026). The ResNet (He et al., 2016) uses the same architecture as that adopted in the hybrid baselines, with 64 hidden channels and 6 residual blocks. For FNO (Li et al., 2021a), we use 16 Fourier modes and a channel width of 64, and adjust the domain-size-related configuration to match the parameter count as closely as possible across datasets. For Transolver Wu et al. (2024), we use a hidden dimension of 64 and 32 slices. Its model capacity is matched by adjusting the number of transformer blocks and the hidden dimension of the MLP layers. These settings are chosen to keep the parameter counts of different baselines comparable, so that the comparison primarily reflects differences in modeling strategy rather than model size.

Table 6: Mean relative L_2 error over the first 60 predicted steps. Model columns identify training viscosity. Lower is better within each row. Best results are shown in bold, and second-best results are underlined.

Test ν	Solver only	HybridOSS (10^{-4})	HybridOSS (10^{-5})
5×10^{-6}	$\underline{0.3216 \pm 0.0158}$	0.3824 ± 0.0153	$\mathbf{0.1516 \pm 0.0154}$
10^{-5}	0.4413 ± 0.0131	$\underline{0.3604 \pm 0.0158}$	$\mathbf{0.1454 \pm 0.0156}$
5×10^{-5}	0.3030 ± 0.0283	$\mathbf{0.2093 \pm 0.0222}$	$\underline{0.2454 \pm 0.0102}$
10^{-4}	$\underline{0.1912 \pm 0.0115}$	$\mathbf{0.1170 \pm 0.0330}$	0.3992 ± 0.0125

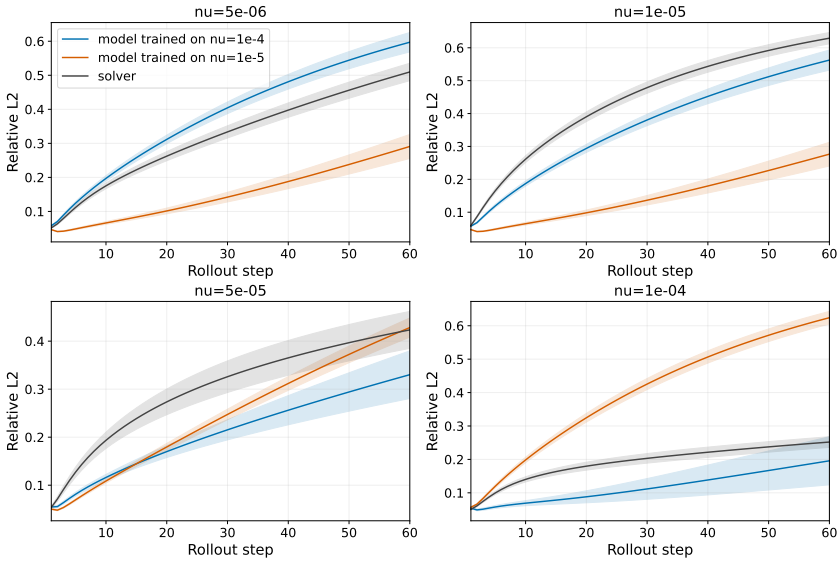


Figure 4: Cross-viscosity generalization of HybridOSS trained at different viscosity conditions.

C Additional Experiments

C.1 Sensitivity to Training Viscosity

We further investigate the influence of the training viscosity by training HybridOSS separately at $\nu = 10^{-5}$ and $\nu = 10^{-4}$, and evaluating both models across all viscosity conditions. Table 6 reports the mean relative L_2 error over the first 60 rollout steps. The results show that each model performs best around its corresponding training regime and can maintain improved accuracy over solver-only for nearby viscosity values. However, when the test viscosity is far from the training condition, the prediction error increases and can exceed that of the solver-only baseline. This trend is observed for models trained at both viscosities.

These results indicate that the learned correction exhibits local generalization across viscosity conditions, while its effectiveness gradually degrades under larger parameter shifts. This suggests that the learned unresolved dynamics remain dependent on the underlying physical regime.