MODEL-AGNOSTIC KNOWLEDGE GUIDED CORRECTION FOR IMPROVED NEURAL SURROGATE ROLLOUT

Anonymous authors
Paper under double-blind review

ABSTRACT

Modeling the evolution of physical systems is critical to many applications in science and engineering. As the evolution of these systems is predominantly governed by partial differential equations (PDEs), there are a number of sophisticated computational simulations which resolve these systems with high accuracy. However, as these simulations incur high computational costs, they are infeasible to be employed for large-scale analysis. A popular alternative to simulators are neural network *surrogates* which are trained in a data-driven manner and are much more computationally efficient. However, these surrogate models suffer from high rollout error when used autoregressively, especially when confronted with training data paucity (i.e., a small number of trajectories to learn from). Existing work proposes to improve surrogate rollout error by either including physical loss terms directly in the optimization of the model or incorporating computational simulators as 'differentiable layers' in the neural network. Both of these approaches have their challenges, with physical loss functions suffering from slow convergence for stiff PDEs and simulator layers requiring gradients which are not always available, especially in legacy simulators. We propose the Hybrid PDE Predictor with Reinforcement Learning (HyPER) model: a model-agnostic, RL based, cost-aware model which combines a neural surrogate, RL decision model, and a physics simulator (with or without gradients) to reduce surrogate rollout error significantly. In addition to reducing rollout error by 34%-96% we show that HyPER learns an intelligent policy that is adaptable to changing physical conditions and resistant to noise corruption.

1 INTRODUCTION

Scientific simulations have long been the workhorse enabling novel discoveries across many scientific disciplines. However, executing fine-grained simulations of a scientific process of interest is a costly undertaking requiring large computational resources and long execution times. In the past decade, the advent of low-cost, efficient GPU architectures has enabled the re-emergence of a powerful function approximation paradigm called deep learning (DL). These powerful DL models, with the ability to represent highly non-linear functions can be leveraged as *surrogates* to costly scientific simulations. Recently, the rapid progress of DL has greatly impacted scientific machine learning (SciML) with the development of neural surrogates in numerous application domains. Some highly-impactful applications include protein structure prediction, molecular discovery Schauperl & Denny (2022); Smith et al. (2018) and domains governed by partial differential equations (PDE) Brunton & Kutz (2024); Raissi et al. (2019); Lu et al. (2021b). Neural surrogates have also been successfully employed for modeling fluid dynamics in laminar regimes like modeling blood flow in cardiovascular systems Kissas et al. (2020) and for modeling turbulent Duraisamy et al. (2019) and multi-phase flows Muralidhar et al. (2021); Raj et al. (2023); Siddani et al. (2021).

Neural Surrogates are Data Hungry. Although neural surrogates are effective at modeling complex functions, this ability is usually conditioned upon learning from a large trove of representative data. This data-hungry nature of popular neural surrogates (like neural operators Li et al. (2020); Lu et al. (2021a)) is well known Tripura et al. (2024); Howard et al. (2023); Lu et al. (2022). However, many scientific applications suffer from *data paucity* due to the high cost of the data collection process (i.e., primarily due to high cost of scientific simulations). Hence, neural surrogates employed to model a scientific process of interest, need to address the data paucity bottleneck by learning effectively with a low volume of training data.

Rollout Errors in Neural Surrogates. Although computational simulations have been designed for modeling various types of physical systems, those exhibiting transient dynamics are especially challenging to model. Solutions to systems exhibiting transient dynamics are usually obtained by discrete-time evolution of the dynamics. Simulators used to model such systems are invoked autoregressively and thereby encounter numerical instability and error buildup over long estimation horizons. Such error buildup during autoregressive invocation is termed *rollout error*. Effective techniques have been developed to reduce rollout error of computational simulations and increase their numerical stability over long rollouts. Although autoregressive rollout of neural surrogates is also affected by rollout error, solutions to minimize this error buildup have not been widely investigated. Recently, (Margazoglou & Magri, 2023) has inspected the stability of echo-state networks during autoregressive rollout and List et al. (2024); Carey et al. (2024); Lippe et al. (2024) has characterized rollout errors in more general neural surrogates. However, a systematic solution to effectively alleviate rollout errors in neural surrogates for modeling transient dynamics is still lacking.

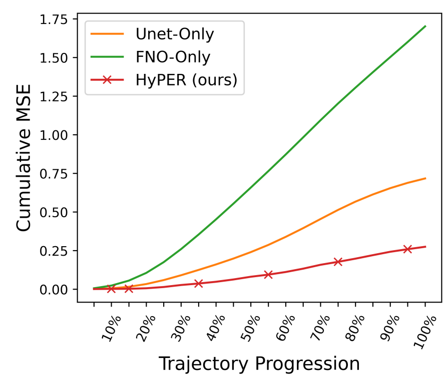

Figure 1: Cumulative MSE depicting *rollout error* for a single trajectory of HyPER vs surrogate only methods. X's mark the timesteps during the trajectory where our RL policy calls the simulator.

Knowledge-Guided Neural Surrogates. One popular method of addressing errors due to data paucity, in neural surrogates is to leverage knowledge of the theoretical model governing the scientific process. Previous efforts have incorporated domain knowledge (in the form of ODEs, PDEs) while training the DL surrogate to develop knowledge-guided learning pipelines (Raissi et al., 2019; Karpatne et al., 2022; Rackauckas et al., 2020; Gao et al., 2021). Most of these approaches incorporate the ODE or PDE governing the system dynamics as soft regularizers (i.e., loss term) while training the neural surrogates. A majority of such approaches have been found to exhibit slow convergence and sometimes catastrophic failures in challenging / stiff PDE conditions (Krishnapriyan et al., 2021; Wang et al., 2022).

Hybrid-Modeling All approaches discussed thus far are so-called *surrogate-only* (SUG) approaches. Here, the computational simulator is employed only as a means of generating data to train the (possibly knowledge-guided) neural surrogate and discarded post the training. SUG approaches employ only the pre-trained neural surrogate during inference. Although SUG provide instantaneous responses relative to computational simulations, they generally have limited generalization ability outside the domain of the training data. An effective complement to SUG approaches are *hybrid-modeling* approaches (Kurz et al., 2022; Karpatne et al., 2022), that jointly resolve a query by incorporating surrogates in conjunction with computational solvers. Otherwise stated, hybrid-modeling pipelines employ a 'solver-in-the-loop' (Um et al., 2020) approach. In addition to neural surrogates, there exist a number of classical hybrid modeling techniques which combine a full order model (FOM) with a reduced order model (ROM) such as proper orthogonal decomposition (Willcox & Peraire, 2002), dynamic model decomposition (Kutz et al., 2016), and multi-scale methods. While these methods are used to accelerate scientific simulations, they are often limited in expressivity, especially in modeling complex non-linear dynamics. Recent hybrid models (Suh et al., 2023) generally have a static coupling between the components in the model and require a static interaction/transition between the full order model (FOM) and the reduced order model (ROM), while our method proposes an adaptable and learnable interaction between the neural surrogate and the simulator. Our proposed method allows dynamic integration of an FOM (fine-grained simulator) with ROM (e.g., neural surrogate or proper orthogonal decomposition) using reinforcement learning.

Knowledge-Guidance with Hybrid-Modeling. Hybrid-modeling approaches are inherently knowledge-guided. A majority of the recent hybrid-modeling approaches are based on directly incorporating PDE solvers as additional layers in the deep learning architecture of neural surro-

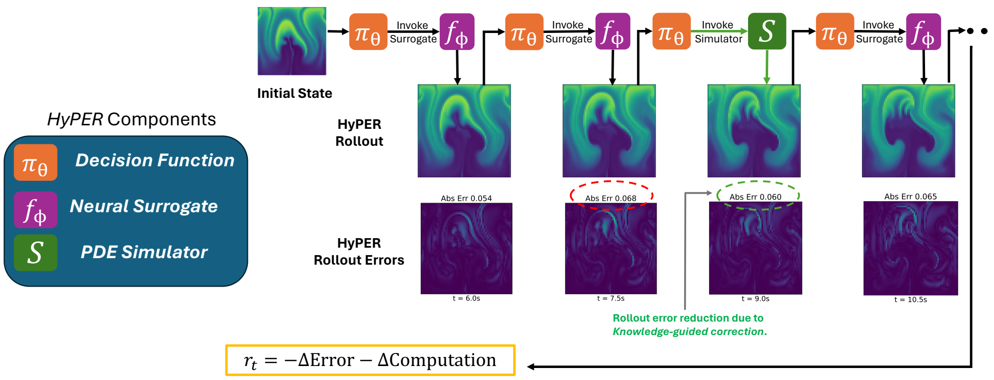

Figure 2: Overview of Hybrid PDE Predictor with RL (HyPER) with example rollout. Here π_θ is the decision model, f_ϕ is the surrogate, and S is the simulator. At $t = 9.0$ in the above trajectory, the decision policy invokes the simulator, correcting the trajectory to reduce rollout error. The effect of this knowledge-guided correction can be observed by a reduction in absolute error (dotted green circle) in the figure.

gates (Chen et al., 2018; Belbute-Peres et al., 2020; Donti et al., 2021; Pachalieva et al., 2022). While such approaches address the issues of large errors under data paucity and during rollout, inherent in SUG approaches, they impose the crippling restriction of *differentiability* on the computational solvers to be incorporated as part of the DL pipeline. Most solvers and computational simulations are NOT differentiable out-of-the-box and hence imposing such differentiability constraints drastically curtails the applicability of current hybrid-modeling approaches.

To address the existing challenges with surrogate-only and hybrid modeling approaches, we propose the **Hybrid PDE Predictor with RL** (HyPER) framework. HyPER is a model-agnostic, simulator-agnostic framework that learns to invoke the costly computational simulator (in a cost-aware manner) as *knowledge-guided correction* to alleviate the effects of rollout errors in surrogates trained with low volumes of training data. Fig. 2 depicts the proposed HyPER framework. Our contributions are as follows.

• HyPER is a first of its kind knowledge-guided correction mechanism that incorporates simulators in the loop without the requirement of the simulators to be *differentiable*.

• HyPER is model agnostic (i.e., functions with any neural surrogate, scientific simulator) and trained in a cost-aware manner, to intelligently invoke the simulator to correct the rollout error of the neural surrogates, only when necessary.

• We demonstrate through rigorous experimentation on in-distribution, out-of-distribution and noisy data that HyPER significantly reduces rollout error relative to SUG approaches by comparing with state-of-the-art neural surrogates.

2 Method

2.1 PDE Prediction

We aim to solve PDEs involving spatial dimensions $\boldsymbol{x} = [x_1, x_2, \ldots, x_m] \in \mathbb{R}^m$ and scalar time $t \in [0, T]$. These PDEs relate solution function $\boldsymbol{u}(\boldsymbol{x}, t) : [0, T] \times \mathbb{R}^m \rightarrow \mathbb{R}^n$ to its partial derivatives over the domain. We assume we have initial conditions $\boldsymbol{u}(\boldsymbol{x}, t = 0)$ and boundary conditions $\boldsymbol{u}_B(\boldsymbol{x} = \boldsymbol{x}_B, t)$ which define the field values at time 0 and at the boundaries of the domain respectively. These time-dependent PDEs can be generally defined as:

$$\frac{\partial \boldsymbol{u}}{\partial t} = \mathcal{F}(\boldsymbol{x}, t, \boldsymbol{u}, \frac{\partial \boldsymbol{u}}{\partial \boldsymbol{x}}, \frac{\partial^2 \boldsymbol{u}}{\partial \boldsymbol{x}^2}, \ldots) \tag{1}$$

We focus on autoregressive *rollout* of these PDEs over time, which can be defined with an function g taking current state and time as inputs and producing the next state:

$$\boldsymbol{u}(\boldsymbol{x}, t + \Delta t) = g(\boldsymbol{u}(\boldsymbol{x}, t), \Delta t) \tag{2}$$

Note that the equation above can be applied autoregressively over any number of timesteps Δt to unroll a PDE trajectory. The cumulative error of this autoregressive process is defined as the *rollout error*, which we aim to minimize.

2.2 HyPER Components

Surrogate ML Model We begin with a machine learning model $f_\phi(\boldsymbol{u}, t)$, which we denote as the surrogate. This model can be any deep learning model with parameters ϕ that predicts next state $\boldsymbol{u}(\boldsymbol{x}, t + \Delta t)$ given starting state and time $\boldsymbol{u}(\boldsymbol{x}, t)$.

Simulator We also define a PDE simulator $S(\boldsymbol{u}, t, \Delta t)$ which numerically solves the PDE to find the next state $\boldsymbol{u}(\boldsymbol{x}, t + \Delta t)$. Crucially, this simulator is only required to return the next state without any gradient information both during training and inference.

Decision Model Finally, we have a decision model $d_\theta(\boldsymbol{u}, \boldsymbol{z}, t)$ which takes current state, conditional features $\boldsymbol{z}$, and current time t and outputs a next action: either call the surrogate f_ϕ or the simulator S. In HyPER we implement our decision model as a learned policy π_θ which we train using reinforcement learning (RL).

We formalize our decision model using a Markov Decision Process (MDP) which is a tuple $(\mathbb{S}, \mathbb{A}, P, r)$. Our states $\mathbb{S}$ consist of current state $\boldsymbol{u}$, conditional features $\boldsymbol{z}$, and current time t. Our action space is binary at each timestep and defined as $a : \{0 = \text{call surrogate}, 1 = \text{call simulator}\}$. Our reward function for a single time-step is:

$$r_t = -\ell(\hat{\boldsymbol{u}}(\boldsymbol{x}, t), \boldsymbol{u}(\boldsymbol{x}, t)) - c(\lambda, k, \tau) \tag{3}$$

Here ℓ represents an *error* function (mean squared error in our case) while $c(\lambda, k, \tau)$ is a *cost* function. τ is the length of the trajectory, $\hat{\boldsymbol{u}}(\boldsymbol{x}, t)$ is the prediction of our model, $\boldsymbol{u}(\boldsymbol{x}, t)$ is the ground truth. The cost function c is computed according to hyperparameter λ, trajectory length τ and number of simulator calls k. The hyperparameter λ is set by the user to specify what percentage of the trajectory to call the simulator. For example, if $\lambda = 0.5$ then the cost function will penalize the reward if the simulator is not called 50% of the time. Our cost function is defined as:

$$c(\lambda, k, \tau) = \left| \frac{k}{\tau} - \lambda \right| \tag{4}$$

Intuitively, reward function 3 optimizes the RL decision model to minimize mean squared error while calling the simulator for λ proportion of the trajectory. To learn a policy $\pi_\theta(\boldsymbol{a}|\boldsymbol{u}, \boldsymbol{z}, t)$ we use the REINFORCE policy gradient algorithm (Sutton et al., 1999) with a baseline in the loss:

$$L_{policy} = \mathbb{E}_t[-\log \pi_\theta(a_t|\hat{\boldsymbol{u}}(\boldsymbol{x}, t), \boldsymbol{z}, t)\text{sg}(r_t - b_t)] \tag{5}$$

Above $\text{sg}(\cdot)$ is stop gradient and b_t is a baseline function which calculates the reward of uniform random actions across the trajectory. The baseline function stabilizes the loss values and leads to better policy learning. This loss increases the probability of actions that have higher advantage when comparing our learned policy to a uniform random policy. For full training details of HyPER see Appendix A.1.

2.3 Experiments

2D Navier Stokes Dataset. We create a 2D incompressible Navier Stokes fluid flow dataset using PhiFlow (Holl & Thuerey, 2024). We generate 1,000 trajectories of 20 timesteps each with a grid size of 64x64. The Navier-Stokes equations in vector velocity form are detailed in Eq. 6.

$$\frac{\partial \boldsymbol{v}}{\partial t} = -\boldsymbol{v} \cdot \nabla \boldsymbol{v} + \mu \nabla^2 \boldsymbol{v} - \nabla p + f; \quad \nabla \cdot \boldsymbol{v} = 0 \tag{6}$$

Eq. 6 comprises a convection term $-\boldsymbol{v} \cdot \nabla \boldsymbol{v}$, diffusion term $\mu \nabla^2 \boldsymbol{v}$ where μ indicates kinematic viscosity, a pressure term ∇p, and external force term f. The velocity divergence term $\nabla \cdot \boldsymbol{v} = 0$ enforces conservation of mass. For our experiments, we generate our dataset with $\Delta t = 1.5s$, buoyancy force $\boldsymbol{f} = [0.0, 0.5]$, and diffusion coefficient $\mu = 0.01$. We set our velocity boundary condition to $\boldsymbol{v} = 0$ (Dirichlet boundary) and our density boundary condition to a Neumann boundary condition $\partial \rho / \partial \boldsymbol{x} = 0$. We generate the trajectory for 20 timesteps at 1.5 seconds per timestep, for a total time of 30 seconds. We split our 1000 trajectories into 4 sets: 400 for surrogate training, 400 for RL training, and 200 for testing.

Subsurface Flow Dataset. To evaluate HyPER's ability to work with different PDEs and problem scales, we generate a subsurface flow dataset using the Julia-based DPFEHM (Pachalieva et al., 2022) simulator. We use DPFEHM to generate a 2D dataset of underground permeability fields with fluid flow modeled by the Richards equation as described below.

$$\frac{\partial \theta}{\partial t} = \nabla \cdot \mathbf{K}(h)(\nabla h - \nabla z) - T^{-1} \tag{7}$$

This equation models the movement of fluid underground in an unsaturated medium. θ represents the volumetric fluid content, $\mathbf{K}(h)$ is the unsaturated hydraulic conductivity, h is the pressure head, ∇z is the geodetic head gradient, and T^{-1} is the fluid sink term. We generate 500 trajectories of 100 timesteps each with a grid size of 50x50 and timestep size of 10 seconds. Out of the 500 trajectories, 200 are used for surrogate training, 200 are used for RL training, and 100 are used for testing. We use Dirchlet boundary conditions (0) for volumetric flux on all sides of our domain but inject fluid at the top center of the domain at a rate of $0.01m/s$ which is pulled down by the force of gravity over time.

3 Results and Discussion

In this section, we investigate the effectiveness of HyPER to model transient PDE systems efficiently and with minimal rollout error compared to surrogate-only (SUG) rollout. We compare with state-of-the-art neural surrogates like U-Net and Fourier Neural Operator (FNO). Specifically, we investigate the effectiveness of HyPER rollouts under changing PDE dynamics and in noisy data settings. Further, we also demonstrate the surrogate-agnostic and simulator-agnostic nature of HyPER. Our experiments seek to answer the following research questions:

- **RQ1:** How effective are HyPER rollouts compared to SUG rollouts for transient PDE systems?
- **RQ2:** Are HyPER rollouts effective under changed physical conditions?
- **RQ3:** Are HyPER rollouts effective under noisy data conditions?
- **RQ4:** How crucial is the intelligent decision model for effective HyPER rollouts?
- **RQ5:** What is the error/efficiency trade-off between SUG, HyPER, and simulator-only paradigms?

3.1 RQ1: HyPER rollouts vs SUG

We begin by comparing HyPER to six SUG baselines: *UNet*, *FNO*, *UNet-Multistep*, *MPP-ZS*, *MPP*, and *PDE-Refiner*. All baselines except UNet-Multistep are trained in a one-step-ahead manner i.e., given the current state of the resolved field they are trained to predict the next state. Mean squared error (MSE) loss is used to train all surrogates over the full resolution trajectory. Note that all these baselines except MPP-ZS are trained with the same dataset that HyPER is trained with. For full training details see Appendix A.1 and for baseline details see Appendix A.4.

Table 1: 2D Navier-Stokes results with best results in bold. Notice that HyPER rollout incurs significantly lower rollout (cumulative) error compared to SUG models.

2D Navier-Stokes	UNet	FNO	UNet-Multistep	MPP-ZS	MPP	PDE-Refiner	HyPER
Final MSE	0.022	0.078	0.016	0.221	0.036	0.023	**0.01**
Cumulative MSE	0.4	1.564	0.273	4.495	0.747	0.406	**0.179**

The UNet model is based on the modern UNet architecture in PDEArena (Gupta & Brandstetter, 2022) while the FNO model is built with the *neuraloperator* library (Kovachki et al., 2021; Li et al., 2020). UNet-Multistep is trained in a scheduled manner to specialize for rollout trajectory prediction, i.e. it minimizes MSE for 1-step ahead prediction, then 2-steps ahead, and so on, up to the full trajectory length. MPP-ZS (zero-shot) and MPP (McCabe et al., 2023) are large transformer based multi-task models which have been pretrained on a variety of fluid prediction tasks. MPP-ZS is the off-the-shelf pretrained model, while MPP is finetuned on our data. PDE-Refiner (Lippe et al., 2024) is a modern diffusion based model which both predicts the next timestep while adding a multi-step noising/denoising process and has been shown to reduce rollout error. Table 1 shows the MSE

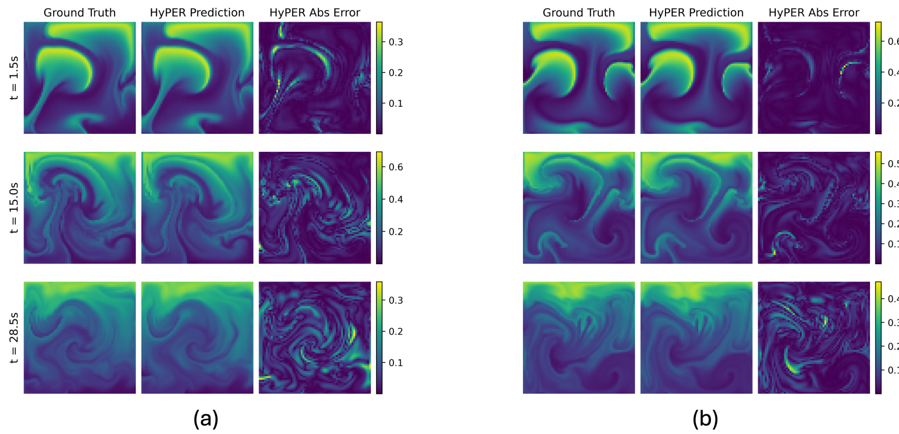

Figure 3: Predictions and absolute error snapshots of HyPER rollout for two distinct trajectories.

of the final trajectory state (Final MSE) as well as the aggregated MSE (Cumulative MSE) over all trajectory states in a rollout. Both metrics are calculated as an average across 200 test trajectories for both SUG and HyPER rollouts. HyPER outperforms all baselines significantly, with an average improvements in cumulative rollout errors (i.e., cumulative MSE) of **67.70%**. By incorporating the simulator, HyPER yields significantly lower cumulative rollout error (relative to SUG approaches) while only invoking the simulator during 30% of the trajectory for knowledge-guided correction of surrogate rollout.

In Figure 1 we plot the rollout MSE performance of HyPER versus the surrogate only (SUG) methods for a single sample trajectory. Note that HyPER outperforms both SUG models by a large margin while only invoking the simulator six times in the 20-step trajectory. Also see Figures 3(a) and 3(b) which depict qualitative snapshots from two distinct HyPER rollouts.

3.2 RQ2: HyPER ROLLOUTS WITH CHANGING PHYSICAL CONDITIONS

Pre-trained neural surrogates are often confronted with modeling contexts with similar PDE dynamics but varied initial / boundary conditions. Effective rollouts under changing physical conditions is hence a crucial requirement for the effective use of neural surrogates in computational science. To this end, we inspect rollout errors of SUG and HyPER, under changing boundary conditions. Specifically, to test the adaptability of HyPER, we investigate HyPER rollouts with changing boundary conditions in our Navier-Stokes experiment. We generate a separate Navier-Stokes dataset (NS changing boundary) which follows the same initial conditions of our previous experiment, but comprises a different velocity boundary condition at the top boundary of the domain. Specifically, the velocity at the top boundary of the domain is increased from 0.0 to 0.5 m/s (imposing an intermittent external forcing effect causing the fluid to escape from the top of the domain) for four intermediate time-steps of the trajectory (timesteps 12-16). Following this, the boundary condition is reverted back to 0.0 for the rest of the trajectory. This results in the fluid escaping out of the top during these time steps changing the PDE dynamics significantly.

Table 2: Results depicting rollout error for 2D Navier-Stokes with changing boundary conditions. We notice that HyPER once again accumulates significantly lower rollout error compared to SUG approaches. This is due to the ability of HyPER to parsimoniously invoke the simulator for knowledge-guided correction to minimize surrogate rollout error propagation, due to the changed boundary condition.

Changing Boundary	UNet-Only	FNO-Only	UNet-FT	FNO-FT	HyPER
Final MSE	0.356	0.541	0.04	0.038	**0.024**
Cumulative MSE	1.859	3.504	1.056	3.453	**0.499**

In this case, surrogates UNet-Only and FNO-Only are not re-trained with the changed boundary condition and hence show poor performance accumulating significant rollout errors. UNet-FT and FNO-FT are fine-tuned on the changing boundary data but still suffer from poor rollout error. We train our HyPER with the intelligent RL policy, a pre-trained surrogate (not trained on to the changed boundary setting) and a simulator (fully aware of the changed boundary condition). The RL policy of HyPER is trained with 400 of these changed trajectories to learn the optimal policy of (frugally) invoking the simulator to apply knowledge-guided correction to reduce surrogate rollout errors. As shown in Table 2, HyPER reduces cumulative error in this scenario by **73.16%** and **85.76%** relative to UNet-Only and FNO-Only rollout errors respectively. Even when compared to fine-tuned surrogates UNet-FT and FNO-FT, HyPER reduces rollout error by **52.75%** and **85.54%** respectively. In Figure 4a we demonstrate that HyPER has much lower rollout error than the UNet-Only and FNO-Only models. Figure 4b shows sample qualitative predictions of HyPER and SUG rollouts under the changing boundary scenario of interest, to further reinforce our point. As illustrated, our model prediction is much closer to the ground truth after the boundary condition has changed and fluid has escaped the box owing to the appropriately invoked knowledge-guided correction by the learned RL policy in response to increasing surrogate rollout error. Fig. 4a shows the significantly lower rollout error for HyPER rollout relative to UNet-Only and FNO-Only models.

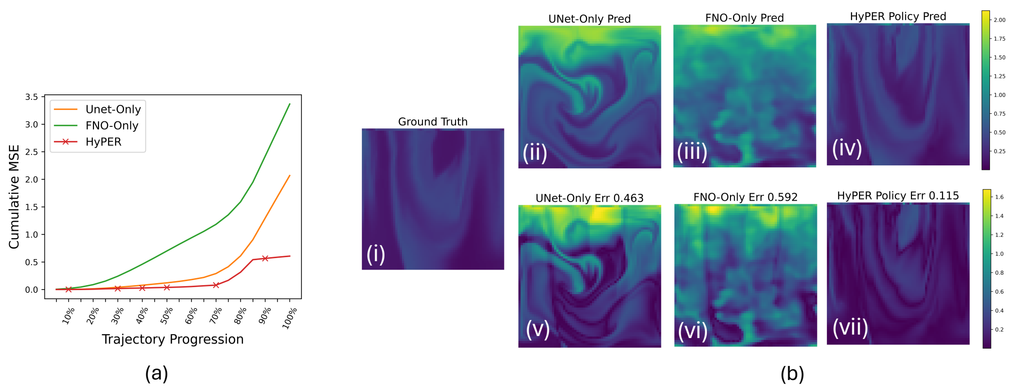

Figure 4: Predictions and absolute error of HyPER vs UNet-Only and FNO-Only for a single trajectory and timestep. Fig. 4a shows the rollout error accumulation over the trajectory with "X" marking the times the simulator is called. Fig. 4b shows the resolved system state for a the same trajectory at a single timestep. Fig. 4b(i) shows the ground truth field while Fig. 4b(ii)-(iv) indicate the result of UNet-Only, FNO-Only and HyPER rollouts respectively. Fig. 4b(v)-(vii) depict the corresponding absolute errors. We notice that only HyPER rollouts capture the correct characteristics relative to the ground truth owing to the knowledge-guided correction while SUG models as well as the random policy rollout are unable to faithfully resolve the trajectory under changing physical conditions.

3.3 RQ3: HyPER rollouts with Noisy Data

Neural surrogates, in addition to being data hungry and accumulating rollout error, have also been known to exhibit catastrophic failures in challenging PDE conditions (e.g., stiff PDEs) Krishnapriyan et al. (2021). As neural surrogates are a crucial part of HyPER rollouts, it is imperative to investigate whether HyPER is capable of *adapting* to such failures by invoking the knowledge-guided correction (i.e., the simulator) to minimize the propagation (and accumulation) of such local failures over the remaining trajectory rollout. Further, another crucial property to investigate is the robustness of the intelligent decision mechanism in HyPER, to noisy, low-quality surrogate predictions. To jointly investigate both goals, we consider a simple (contrived) context with noisy inputs supplied to the HyPER RL policy. This additive noise, injected at specific steps to *corrupt* the surrogate output in the trajectory, serves to mimic low-quality surrogate predictions. Hence, experiments with such noisy inputs help characterize the ability of HyPER to adapt to sudden local changes during rollout (like catastrophic surrogate failure) in addition to demonstrating its ability for robust decision-making under noisy data conditions.

To carry out this investigation, we add random Gaussian noise with mean 0 at four separate variance (σ^2) scales at fixed timesteps of our trajectory. The surrogate models are never trained with this noisy data and therefore perform poorly when encountering it. We train the RL policy of HyPER

Table 3: Results depicting rollout error for the 2D-Navier-Stokes experiment, with random Gaussian noise added to inputs, at timesteps 12-16. σ^2 is the *scale* of the noise. The percentage reduction of cumulative MSE rollout error by HyPER, over the best performing SUG model is in parentheses.

Experiment	Final MSE			Cumulative MSE		
	UNet-Only	FNO-Only	HyPER	UNet-Only	FNO-Only	HyPER
Unimodal $\sigma^2 = 1.0$	0.278	0.178	**0.075**	2.537	3.445	**1.164** (54.12%)
Unimodal $\sigma^2 = 0.75$	0.151	0.161	**0.05**	1.653	2.798	**0.756** (54.27%)
Unimodal $\sigma^2 = 0.5$	0.075	0.145	**0.033**	1.023	2.319	**0.465** (54.54%)
Unimodal $\sigma^2 = 0.25$	0.044	0.134	**0.02**	0.664	2.016	**0.275** (58.58%)

with a small set (400) of these noisy trajectories while keeping our surrogate model static. We test two different noise corruption scenarios added to a 20 step trajectory rollout. (i) unimodal noise: a case where noise is added at timesteps [12-16] and (ii) bimodal noise: a more sophisticated case where noise is added at two different time windows of [2-4] and [15-16]. The unimodal noise results are presented in Table 3, where we see that HyPER rollouts outperform both state-of-the-art SUG approaches. In this case we notice a reduction in cumulative MSE by **54.12%-58.58%** across the four noise scales. We can further see that unlike SUG approaches, HyPER rollouts degrade gracefully with increasing noise scale. This is only possible due to HyPER appropriately invoking the simulator for knowledge-guided correction exactly at (or very close to) the regions in the rollout subjected to data corruption. Note that as the noise scale σ^2 decreases, the gap between HyPER and the baselines decreases, but even in the lowest noise case of $\sigma^2 = 0.25$, HyPER significantly reduces rollout MSE by an average of 58.58%.

We see a similar behavior in the case of bimodal noise (See. Appendix A.2 for bimodal noise results) with a reduction in cumulative MSE error of **44.01%-76.49%** across the four noise scales. We believe this improved performance in a more complex noise distribution is a result of our RL policy learning the more complex error distribution while SUG methods accumulate larger error over the two different noise windows.

3.4 RQ4: Investigating HyPER Rollouts vs. Random Policy Rollouts

To evaluate whether HyPER learns an effective RL policy, we compare it to a Random Policy baseline. This baseline is designed to invoke the simulator the same number of times as the HyPER RL policy for a particular rollout, but with uniform random probability over each timestep. By comparing the HyPER policy rollout to a Random Policy rollout with the same budget, we show that HyPER learns a superior performing policy in our experimental scenarios and that learning an intelligent RL policy is indeed necessary.

Table 4: HyPER versus a random policy which calls the simulator the same number of times.

Experiment	Final MSE		Cumulative MSE		Cumulative Wins %
	Random Policy	HyPER	Random Policy	HyPER	HyPER
Noise Free	0.011	**0.01**	0.189	**0.179**	56.00%
Bimodal $\sigma^2 = 1.0$	0.222	**0.103**	2.768	**1.972**	69.70%
Bimodal $\sigma^2 = 0.75$	0.067	**0.042**	1.431	**1.038**	72.60%
Bimodal $\sigma^2 = 0.5$	0.029	**0.016**	0.716	**0.443**	79.00%
Bimodal $\sigma^2 = 0.25$	0.021	**0.019**	0.482	**0.433**	56.25%
Changing Boundary	0.147	**0.024**	0.854	**0.499**	77.35%

We summarize these results in Table 4, which also shows the percentage of trajectories over which HyPER reduces MSE compared to Random Policy (Cumulative Wins %). A 'win' is charactereized by a HyPER rollout that yields a lower cumulative MSE than the corresponding Random Policy rollout. In the 'Noise Free' trajectories, HyPER only wins for 56% of the trajectories because the MSEs at each timestep in the Noise Free case have low variance, so a uniform Random Policy performs fairly well. This is also true in the bimodal noise $\sigma = 0.25$ experiment as lower-noise scales lead to relatively slower rollout error accumulation and hence invoking the simulator uniformly somewhere around the corrupted time windows ([2-4], [15-16]) suffices to correct the errors, resulting in reasonably good performance for the Random Policy. However, in the higher bimodal noise scales (with fast and large rollout error accumulation), precise invocation of the simulator for knowledge-guided correction is imperative to prevent error accumulation. Hence, we see HyPER owing to its intelligent (RL-based) decision policy, has a higher percentage of wins ($\approx 70\%$) and significantly lower cumulative MSE at higher noise scales. We also see the strength of HyPER's learned policy when considering the changing boundary condition experiment which has a 77.35 win percentage over Random Policy and reduces cumulative MSE by 41.57%. This result demonstrates the effectiveness of HyPER in realistic physical scenarios and noisy conditions.

3.5 RQ5: Cost versus Accuracy Trade-Off

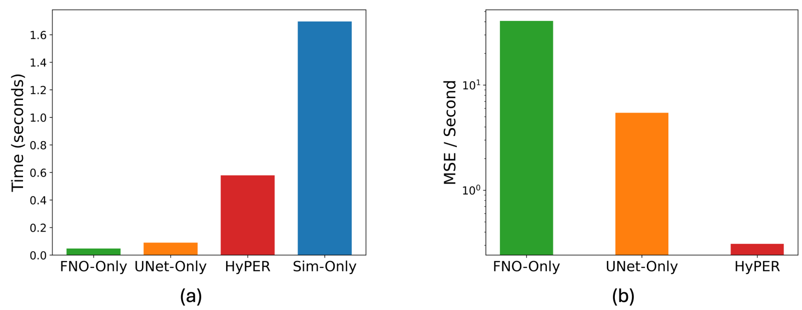

Figure 5: Figure 5(a) shows the average time of PDE prediction of a full trajectory for each method. Figure 5(b) illustrates the error per unit time (*lower is better*) for each method. We do not show the Sim-Only case here as we assume error is effectively zero for the simulator.

A natural question to ask is how much of a cost we pay in wall clock time when utilizing HyPER compared to baselines? We evaluate this by measuring the average time of each method on our 200 test trajectories and show results in Figure 5(a). While the wall clock time of SUG methods is lower than HyPER, our method has much lower rollout error, which we illustrate in Figure 5(b). Here we show the error per unit time (*lower is better*) and we see that HyPER has lower rollout error per second by a large margin in comparison to the baselines (note this plot is on a log scale). While calling the simulator incurs a time cost, the user can specify the λ parameter to adjust to their requirements allowing HyPER to be flexible in varying settings.

3.6 Surrogate-Agnostic and Simulator-Agnostic Design of HyPER

To demonstrate the model-agnostic capability of HyPER, we train it on the subsurface flow task (dataset details in Sec. 2.3), comprising longer (i.e., 100 step) trajectories. In this case, we train HyPER using the Julia simulator DPFEHM, which simulates underground fluid flow in porous media. As we see in Table 5, HyPER outperforms both SUG baselines by a cumulative MSE per timestep

Table 5: 2D Subsurface flow experiment results for a 100 timestep trajectory.

Experiment	Cumulative MSE per timestep		
	UNet-Only	FNO-Only	HyPER
Subsurface	4.345	10.938	**0.271**

reduction of 93.76% and 97.52%. The integration of a distinct Julia-based simulator in a scenario with larger physical scales and times, shows that HyPER is surrogate and simulator agnostic and can perform well and reduce rollout error in multiple problem settings.

4 Related Work

Deep learning (DL) surrogates have been broadly employed in two paradigms to accelerate scientific discovery namely, surrogate-only (SUG) and hybrid-modeling paradigms.

SUG approaches fully circumvent the use of computational simulations during inference and only employ simulations to generate training data. The popular U-Net (Ronneberger et al., 2015) model owing to its ability to capture spatial and temporal dynamics at multiple scales is a notable SUG architecture. The U-net model has recently (Gupta & Brandstetter, 2022) demonstrated state of the art performance on various fluid dynamics benchmarks. Operator learning (Kovachki et al., 2021) approaches that learn function families of PDEs rather than single PDE instances have also been investigated to be effective neural surrogates. Two notable operator learning models are the deep operator network (Lu et al., 2021a) and the Fourier neural operator (FNO) (Li et al., 2020) models. Multiple investigations have been carried out employing operator learning techniques including extending them to multi-resolution (Howard et al., 2023; Lu et al., 2022) settings. Recently Takamoto et al. (2022) have also demonstrated that FNOs yield state-of-the-art results on benchmark tasks.

Knowledge-guided SUG approaches like the popular Physics-informed neural network (PINN) (Raissi et al., 2019; Jagtap & Karniadakis, 2020; Cuomo et al., 2022) have also been effectively employed for improved generalization. A related paradigm of Universal Differential Equations (Rackauckas et al., 2020) utilizes surrogates to estimate (in a data-driven manner) focused sub-components of governing equations for better process modeling. Such approaches assume end-to-end gradient based training in a physics-informed manner and are known to converge slowly and converge to trivial solutions under data paucity and stiff PDE conditions Krishnapriyan et al. (2021) owing to catastrophic gradient imbalance Wang et al. (2021) between data-driven and physics-guided loss terms.

SUG for Transient PDE Dynamics. All SUG approaches struggle to model transient PDE systems in an autoregressive manner and incur rollout error (Carey et al., 2024). In the work of Lippe et al. (2024), it is demonstrated how the spectral bias of traditional neural surrogates leads to significant error accumulation and they propose an initial *refinement* solution inspired by the process in diffusion modeling. Separately, List et al. (2024) have conducted a characterization of SUG rollout error and comment about the significant improvement obtainable by incorporating simulators in-the-loop.

Hybrid-modeling for Transient PDE Dynamics. Complementary to SUG approaches, hybrid-modeling approaches retain the simulation and resolve each query employing the neural surrogate and the simulator 'in-the-loop'. However, a major drawback with such approaches (Chen et al., 2018; Belbute-Peres et al., 2020; Um et al., 2020; Donti et al., 2021; Pachalieva et al., 2022) is that they require the simulators to be *differentiable* as they are mostly employed as additional layers in the neural network architectures to be trained end-to-end with the neural surrogates.

5 Conclusion and Future Work

This work presents a first of its kind knowledge-guided correction mechanism to reduce rollout errors in neural surrogates that model transient PDE systems. Specifically, our proposed method Hybrid PDE Predictor with RL (HyPER) learns reinforcement learning based cost-aware control policies to parsimoniously invoke (costly) simulation steps to *correct* erroneous surrogate predictions. In contrast to existing approaches that employ simulators 'in-the-loop' with neural surrogates, HyPER does not impose any differentiablity restrictions on the computational simulations. Further, HyPER is surrogate and simulator agnostic and is designed to be applicable to any neural surrogate and off-the-shelf simulator capable of resolving transient PDE systems.

We have demonstrated the effectiveness of our proposed HyPER model in traditional in-distribution rollouts, under changing physical conditions and under noisy data conditions. In each case, we demonstrate that HyPER yields significantly lower rollout errors (with parsimonious invocation of the costly simulation step), than surrogate-only rollouts of state-of-the-art neural surrogates. Overall HyPER yields significant improvements of cumulative rollout error over state-of-the-art surrogate-only approaches with **67.70**% improvement for in-distribution rollouts, **79.46**% improvement for rollouts under changing physical conditions and **58.28**% improvement for rollouts under noisy data conditions. In the future, we will investigate more sophisticated RL policies based in the actor-critic paradigm to further improve the sample efficiency of HyPER. We will also explore extensions of HyPER to more challenging problems in multi-physics contexts as well as multi-phase flows.

References

Filipe De Avila Belbute-Peres, Thomas Economon, and Zico Kolter. Combining differentiable pde solvers and graph neural networks for fluid flow prediction. In *international conference on ma-*

chine learning, pp. 2402–2411. PMLR, 2020.

Steven L Brunton and J Nathan Kutz. Promising directions of machine learning for partial differential equations. *Nature Computational Science*, 4(7):483–494, 2024.

N Carey, L Zanisi, S Pamela, V Gopakumar, J Omotani, J Buchanan, and J Brandstetter. Data efficiency and long term prediction capabilities for neural operator surrogate models of core and edge plasma codes. *arXiv preprint arXiv:2402.08561*, 2024.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

Salvatore Cuomo, Vincenzo Schiano Di Cola, Fabio Giampaolo, Gianluigi Rozza, Maziar Raissi, and Francesco Piccialli. Scientific machine learning through physics–informed neural networks: Where we are and what's next. *Journal of Scientific Computing*, 92(3):88, 2022.

Priya L Donti, David Rolnick, and J Zico Kolter. Dc3: A learning method for optimization with hard constraints. *arXiv preprint arXiv:2104.12225*, 2021.

Karthik Duraisamy, Gianluca Iaccarino, and Heng Xiao. Turbulence modeling in the age of data. *Annual review of fluid mechanics*, 51(1):357–377, 2019.

Han Gao, Luning Sun, and Jian-Xun Wang. Phygeonet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state pdes on irregular domain. *Journal of Computational Physics*, 428:110079, 2021.

Jayesh K Gupta and Johannes Brandstetter. Towards multi-spatiotemporal-scale generalized pde modeling. *arXiv preprint arXiv:2209.15616*, 2022.

Philipp Holl and Nils Thuerey. Φ_{flow} (PhiFlow): Differentiable simulations for pytorch, tensorflow and jax. In *International Conference on Machine Learning*. PMLR, 2024.

Amanda A Howard, Mauro Perego, George Em Karniadakis, and Panos Stinis. Multifidelity deep operator networks for data-driven and physics-informed problems. *Journal of Computational Physics*, 493:112462, 2023.

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

Anuj Karpatne, Ramakrishnan Kannan, and Vipin Kumar. *Knowledge Guided Machine Learning: Accelerating Discovery Using Scientific Knowledge and Data*. CRC Press, 2022.

Georgios Kissas, Yibo Yang, Eileen Hwuang, Walter R Witschey, John A Detre, and Paris Perdikaris. Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4d flow mri data using physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 358:112623, 2020.

Nikola B. Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew M. Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces. *CoRR*, abs/2108.08481, 2021.

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

Stefan Kurz, Herbert De Gersem, Armin Galetzka, Andreas Klaedtke, Melvin Liebsch, Dimitrios Loukrezis, Stephan Russenschuck, and Manuel Schmidt. Hybrid modeling: towards the next level of scientific computing in engineering. *Journal of Mathematics in Industry*, 12(1):8, 2022.

J Nathan Kutz, Steven L Brunton, Bingni W Brunton, and Joshua L Proctor. *Dynamic mode decomposition: data-driven modeling of complex systems*. SIAM, 2016.

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

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, 2024.

Bjoern List, Li-Wei Chen, Kartik Bali, and Nils Thuerey. How temporal unrolling supports neural physics simulators. *arXiv preprint arXiv:2402.12971*, 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, 2021a.

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

Lu Lu, Raphaël Pestourie, Steven G Johnson, and Giuseppe Romano. Multifidelity deep neural operators for efficient learning of partial differential equations with application to fast inverse design of nanoscale heat transport. *Physical Review Research*, 4(2):023210, 2022.

Georgios Margazoglou and Luca Magri. Stability analysis of chaotic systems from data. *Nonlinear Dynamics*, 111(9):8799–8819, 2023.

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.

Nikhil Muralidhar, Jie Bu, Ze Cao, Neil Raj, Naren Ramakrishnan, Danesh Tafti, and Anuj Karpatne. Phyflow: Physics-guided deep learning for generating interpretable 3d flow fields. In *2021 IEEE International Conference on Data Mining (ICDM)*, pp. 1246–1251. IEEE, 2021.

Aleksandra Pachalieva, Daniel O'Malley, Dylan Robert Harp, and Hari Viswanathan. Physics-informed machine learning with differentiable programming for heterogeneous underground reservoir pressure management. *Scientific Reports*, 12(1):18734, 2022.

Christopher Rackauckas, Yingbo Ma, Julius Martensen, Collin Warner, Kirill Zubov, Rohit Supekar, Dominic Skinner, Ali Ramadhan, and Alan Edelman. Universal differential equations for scientific machine learning. *arXiv preprint arXiv:2001.04385*, 2020.

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

Neil Ashwin Raj, Danesh Tafti, and Nikhil Muralidhar. Comparison of reduced order models based on dynamic mode decomposition and deep learning for predicting chaotic flow in a random arrangement of cylinders. *Physics of Fluids*, 35(7), 2023.

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

Michael Schauperl and Rajiah Aldrin Denny. Ai-based protein structure prediction in drug discovery: Impacts and challenges. *Journal of Chemical Information and Modeling*, 62(13):3142–3156, 2022.

Bhargav Siddani, S Balachandar, William C Moore, Yunchao Yang, and Ruogu Fang. Machine learning for physics-informed generation of dispersed multiphase flow using generative adversarial networks. *Theoretical and Computational Fluid Dynamics*, 35:807–830, 2021.

Justin S Smith, Adrian E Roitberg, and Olexandr Isayev. Transforming computational drug discovery with machine learning and ai, 2018.

Seung Won Suh, Seung Whan Chung, Peer-Timo Bremer, and Youngsoo Choi. Accelerating flow simulations using online dynamic mode decomposition. *arXiv preprint arXiv:2311.18715*, 2023.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

Makoto Takamoto, Timothy Praditia, Raphael Leiteritz, Daniel MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. Pdebench: An extensive benchmark for scientific machine learning. *Advances in Neural Information Processing Systems*, 35:1596–1611, 2022.

Tapas Tripura, Akshay Thakur, and Souvik Chakraborty. Multi-fidelity wavelet neural operator surrogate model for time-independent and time-dependent reliability analysis. *Probabilistic Engineering Mechanics*, 77:103672, 2024.

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. *Advances in Neural Information Processing Systems*, 33:6111–6122, 2020.

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

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

Karen Willcox and Jaime Peraire. Balanced model reduction via the proper orthogonal decomposition. *AIAA journal*, 40(11):2323–2330, 2002.

A Appendix

A.1 HyPER Training Procedure

Below we detail the RL training procedure of HyPER.

Algorithm 1 HyPER Training Algorithm

Require: Dataset: $\mathcal{D}$, Pretrained Surrogate Model: f_ϕ, Simulator: S, Decision Model: d_θ, Trajectory length: τ, Simulator proportion hyperparameter: λ, Learning rate: η, Error function (MSE in our case): ℓ, Cost function: c

for $\boldsymbol{u}, \boldsymbol{z} \in \mathcal{D}$ **do** ▷ For every trajectory in dataset, get field $\boldsymbol{u}$ and conditional features $\boldsymbol{z}$
  $k \leftarrow 0$ ▷ Initialize number of simulator calls
  $R_d \leftarrow [\,]$ ▷ Initialize decision model rewards list
  $R_b \leftarrow [\,]$ ▷ Initialize baseline rewards list
  $L \leftarrow [\,]$ ▷ List to store log probabilities of each action
  $\hat{\boldsymbol{u}}(\boldsymbol{x}, -1) \leftarrow 0$ ▷ Set initial field prediction to 0
  for $t \in [0, \tau]$ **do** ▷ For every time-step in trajectory
    $p \leftarrow d_\theta(\hat{\boldsymbol{u}}(\boldsymbol{x}, t-1), \boldsymbol{z}, t)$ ▷ Get RL model probabilities for next action
    $a \sim \text{Bernoulli}(p)$ ▷ Sample next action
    $L \mathrel{+}= a \log(p) + (1-a)\log(1-p)$ ▷ Store log probability of action
    if $a = 0$ **then**
      $\hat{\boldsymbol{u}}(\boldsymbol{x}, t) \leftarrow f_\phi(\hat{\boldsymbol{u}}(\boldsymbol{x}, t-1), t)$ ▷ Call surrogate for next step prediction
    else if $a = 1$ **then**
      $\hat{\boldsymbol{u}}(\boldsymbol{x}, t) \leftarrow S(\hat{\boldsymbol{u}}(\boldsymbol{x}, t-1), t)$ ▷ Call simulator for next step prediction
      $k \mathrel{+}= 1$ ▷ Track number of times simulator called
    end if
    $R_d \mathrel{+}= -\ell(\hat{\boldsymbol{u}}(\boldsymbol{x}, t), \boldsymbol{u}(\boldsymbol{x}, t)) - c(\lambda, k, \tau)$ ▷ Store policy reward based on MSE and cost function
  end for
  $I \sim \text{UniformWithoutReplacement}([0, \tau], k)$ ▷ Sample k times between $[0, \tau]$ without replacement, this list will contain the time-steps at which the random baseline will call the simulator
  for $t \in [0, \tau]$ **do** ▷ Run random baseline for trajectory
    if $t \notin I$ **then**
      $\hat{\boldsymbol{u}}(\boldsymbol{x}, t) \leftarrow f_\phi(\hat{\boldsymbol{u}}(\boldsymbol{x}, t-1), t)$ ▷ Call surrogate for next step prediction
    else if $t \in I$ **then**
      $\hat{\boldsymbol{u}}(\boldsymbol{x}, t) \leftarrow S(\hat{\boldsymbol{u}}(\boldsymbol{x}, t-1), t)$ ▷ Call simulator for next step prediction
    end if
    $R_b \mathrel{+}= -\ell(\hat{\boldsymbol{u}}(\boldsymbol{x}, t), \boldsymbol{u}(\boldsymbol{x}, t)) - c(\lambda, k, \tau)$ ▷ Store baseline reward using MSE and cost function
  end for
  $\nabla_\theta J \leftarrow -\nabla_\theta L \cdot \text{StopGradient}(R_d - R_b)$ ▷ Calculate policy gradient, this is done element-wise and then summed
  $\theta \leftarrow \theta - \eta \nabla_\theta J$ ▷ Update RL decision model parameters
end for

A.2 RQ3: HyPER Rollouts with Noise Data: Bimodal Noise Experiments

Table 6 depicts the performance of HyPER in the bimodal noise case. We notice results similar to the unimodal noise case, with a reduction in cumulative MSE error of **44.01%-76.49%**. We believe this improved performance in a more complex noise distribution is a result of our RL policy learning the more complex error distribution while the SUG methods accumulate larger error over two different noise windows.

Table 6: Results depicting rollout error for the 2D-Navier-Stokes experiment with random Gaussian noise added at timesteps 2-4, 15-16. σ^2 is the *scale* of the noise. The percentage reduction of cumulative MSE rollout error by HyPER, over the best performing SUG model is in parentheses.

Experiment	Final MSE			Cumulative MSE		
	UNet-Only	FNO-Only	HyPER	UNet-Only	FNO-Only	HyPER
Bimodal $\sigma^2 = 1.0$	4.954	0.154	**0.103**	16.663	3.522	**1.972** (44.01%)
Bimodal $\sigma^2 = 0.75$	0.869	0.142	**0.042**	4.739	2.812	**1.038** (63.09%)
Bimodal $\sigma^2 = 0.5$	0.095	0.136	**0.016**	1.884	2.309	**0.443** (76.49%)
Bimodal $\sigma^2 = 0.25$	0.045	0.129	**0.019**	1.114	2.0	**0.433** (61.13%)

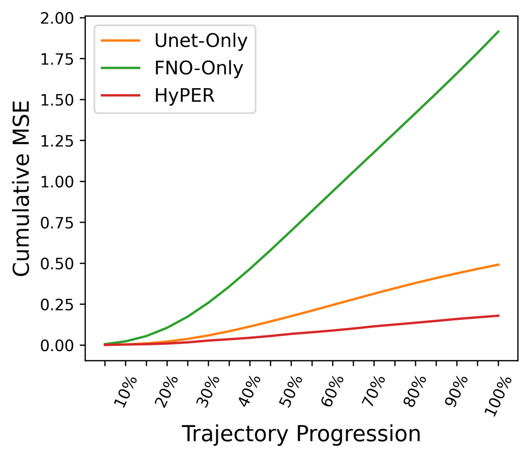

Figure 6: Average Cumulative MSE over all test trajectories of HyPER vs. UNet-Only and FNO-Only.

A.3 HyPER Average Performance

Figure 6 shows the rollout error of HyPER vs SUG methods over all 200 Navier-Stokes test trajectories. This demonstrates that HyPER is effective in reducing rollout error significantly over a large set of unseen test trajectories.

A.4 Model Architectures and Parameters

A.4.1 Model Training Details

We build and train all our neural surrogate models using the PyTorch library on a single Nvidia RTX A6000 GPU. The PhiFlow simulator used in the Navier-Stokes experiment runs on the same GPU. The DPFEHM Julia simulator runs using multi-threading on an Intel(R) Xeon(R) Platinum 8358 CPU @ 2.60GHz.

The UNet, FNO, UNet-FT, FNO-FT, UNet-Multistep, MPP, and PDE-Refiner baselines are trained with same number of samples as HyPER for the sake of fair comparison. The UNet-Only and FNO-Only baseline models are trained with a smaller dataset of 400 trajectories which are separate from the 400 trajectories HyPER's RL policy is trained with. UNet-Multistep, MPP, and PDE-Refiner are trained for 200 epochs so we could compare with strong surrogate baselines while UNet, FNO, UNet-Only, FNO-Only, UNet-FT, FNO-FT were trained for 50 epochs to evaluate results for simpler models.

We only train the HyPER RL ResNet model for 30 epochs to demonstrate that HyPER does not require extensive training to outperform the baselines.

- UNet-Only: 1.05 hours (50 epochs, 400 trajectories)

- FNO-Only: 0.50 hours (50 epochs, 400 trajectories)
- UNet: 2.12 hours (50 epochs, 800 trajectories)
- FNO: 0.89 hours (50 epochs, 800 trajectories)
- UNet-Multistep: 9.0 hours (200 epochs, 800 trajectories)
- MPP: 9.58 hours (200 epochs, 800 trajectories)
- PDE-Refiner: 3.66 hours (200 epochs, 800 trajectories)
- HyPER: 5.21 hours (30 epochs)

A.4.2 Model Parameters

Our UNet model is built on top of the PDEArena modern UNet architecture with wide residual blocks and training parameters in Table 7.

Table 7: UNet parameters.

Parameter	Value
Model Size (# parameters)	12,295,233
Hidden Channels	64
Activation Function	GELU
Channel Multipliers	[1, 2, 2]
Num Residual Blocks Per Channel	2
Sinusoidal Time Embedding	Yes
Learning Rate	1e-4
Optimizer	Adam
Loss Function	MSE
Epochs	50

Our FNO model is built using the neuraloperator library and is constructed to have a similar number of parameters as our UNet for a fair comparison. See Table 8 for details.

Table 8: FNO parameters.

Parameter	Value
Model Size (# parameters)	12,437,057
Number of Fourier Modes	27
Hidden Channels	64
Lifting Channels	256
Projection Channels	256
Learning Rate	1e-5
Optimizer	Adam
Loss Function	MSE
Epochs	50

The HyPER RL model is a lightly modified version of the ResNet34 model from the Torchvision library. See Table 9 for parameters.

The Multiple Physics Pretrained model (MPP) is adapted from MPP github. Note that we train this model for 200 epochs while UNet and FNO are only trained for 50. See Table 10 for details.

The PDE-Refiner model was adapted from PDE-Refiner to work with our UNet model. Note that we train this model for 200 epochs while UNet and FNO are only trained for 50. See Table 11 for details.

Table 9: RL ResNet parameters.

Parameter	Value
Model Size (# parameters)	11,751,300
Number of Layers	[3, 4, 6, 3]
Hidden Channels	64
Sinusoidal. Time Embedding	Yes
Activation Function	ReLU
Learning Rate	1e-5
Optimizer	Adam
Reward Function	Equation 3
Epochs	30

Table 10: Multiple Physics Pretraining parameters.

Parameter	Value
Model Size (# parameters)	28,979,436
Patch Size	16x16
Embedding Dimension	384
Number of Axial Attention Heads	6
Number of Transformer Blocks	12
Epochs	200

Table 11: PDE-Refiner parameters.

Parameter	Value
Model Size (# parameters)	12,378,241
Number of Refinement/Denoising Steps	3
Minimum Noise Scale	4e-7
Hidden Channels	64
Activation Function	GELU
Channel Multipliers	[1, 2, 2]
Num Residual Blocks Per Channel	2
Sinusoidal Time Embedding	Yes
Learning Rate	1e-4
Optimizer	Adam
Loss Function	MSE
Epochs	200