PhysGuard: Fisher-Guided Gradient Projection for Sim-to-Real Neural PDE Surrogates

Changjian Zhou[1] * **Junfeng Fang**[2] **Negin Yousefpour**[1] † **Peng Wu**[3]
Bin Yan[1] **Guillermo A. Narsilio**[1]

[1]Faculty of Engineering and IT, University of Melbourne
[2]School of Computing, National University of Singapore
[3]Artificial Intelligence Research Institute, IFLYTEK Co., Ltd.

Abstract

Neural operator models trained on simulation data often lose accuracy when applied to experimental measurements due to the *sim-to-real gap*. Standard fine-tuning with limited real data can reduce this gap, but it may also damage the core physics-relevant representations learned during pretraining. Although knowledge-preserving adaptation has been widely investigated in vision or language tasks, it remains unclear whether these methods are suitable for neural operators whose architectures and protected knowledge are fundamentally different. Neural operators need to preserve core-scale physical structures rather than semantic or visual features. We propose PhysGuard, a physics-preserving framework for accurate sim-to-real adaptation of neural operators. Specifically, PhysGuard uses the empirical Fisher Information Matrix computed on simulation data to identify physics-critical parameter directions, then restricts fine-tuning updates to directions that do not interfere with them. A layer-wise Gram-matrix formulation makes this efficient for models with millions of parameters, while an adaptive threshold automatically determines the protected subspace size. A spectral probe experiment shows that the dominant Fisher directions are strongly associated with low-frequency output structures. Experiments on RealPDEBench across four neural operator architectures and different physical systems show that PhysGuard performs strongly on most evaluation metrics compared to baselines. The benefits are most evident under severe domain shift, where it reduces low-frequency error by up to 32% compared to standard fine-tuning while maintaining adaptability. Our code is available at https://github.com/Data-Driven-Computational-Geotechnics/PhysGuard.

1 Introduction

Training on simulations and deploying in the real world has become a standard workflow in scientific machine learning (SciML) [1, 2]. Neural operator models have emerged as a particularly promising class of neural PDE surrogates within this paradigm [3], learning solution mappings from large-scale simulation data and enabling rapid inference for complex physical systems [4, 5]. Yet their performance often degrades substantially when applied to real experimental measurements, a phenomenon known as the sim-to-real gap [6, 7], as shown in Figure 1. This gap stems from the idealized assumptions that underlie the simulations, whereas the experimental data exhibit measurement noise, unmodeled effects, and other sources of variability. Consequently, neural operators trained exclusively on simulation data may fail to generalize to real-world scenarios, limiting their practical effectiveness in engineering applications.

*Email: changjianz@student.unimelb.edu.au.
†Corresponding author. Email: negin.yousefpour@unimelb.edu.au.

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

Fine-tuning on limited real data is the most direct solution [8], but it comes with an important risk. Current studies have shown that unconstrained fine-tuning can damage the low-frequency physical structures learned during pretraining [9, 10]. Specifically, during optimization, the model may focus on fitting high-frequency noise while neglecting large-scale coherent patterns such as vortex streets and mean flow profiles [11], which often capture the core physics of the system.

A natural approach is to regularize the fine-tuning objective, for instance, by penalizing deviations from the pretrained weights [12]. However, such penalties apply uniformly across all parameters and are sensitive to the choice of regularization strength. More structured alternatives have proven effective for vision and language tasks, including importance-weighted regularization (EWC [13]), gradient projection (GPM [14]), and parameter-efficient fine-tuning (LoRA [15]). Yet, neural operators differ substantially from these models in both architecture and learning objective. Rather than encoding semantic or class-level features, they rely on mechanisms such as spectral convolutions [16], branch-trunk factorisations [17], and physics-aware attention [18] to preserve governing PDE physics. Whether these techniques transfer to the neural operator setting has not been systematically investigated.

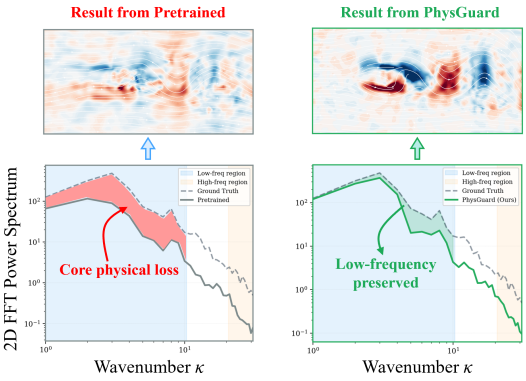


Figure 1: Predicted fields (top) and power spectra (bottom) for the simulation-pretrained model (left) and the same model after PhysGuard fine-tuning on real data (right). Dashed curves show the ground truth. PhysGuard better matches the ground-truth spectrum at low frequencies. Best viewed in color.

We propose **PhysGuard**, a physics-preserving framework for sim-to-real adaptation of neural operators. The key insight is that physical knowledge in a pretrained neural operator concentrates along a small number of principal directions in parameter space (empirically characterized in Section 4.2). Building on this, PhysGuard computes the empirical Fisher Information Matrix (FIM) on simulation data and projects fine-tuning gradients onto the null space of its top eigenvectors. This blocks updates that would overwrite physics-critical parameters while leaving all other directions free for adaptation. The method introduces no auxiliary loss terms, and applies to any differentiable architecture. We validate it on the RealPDEBench [6] across four architectures and three physical scenarios. Our contributions are:

- We propose PhysGuard, a framework for sim-to-real adaptation of neural operators that preserves pretrained physics. A Gram-matrix kernel makes the subspace estimation tractable for million-parameter models.
- Through a spectral probe, we reveal that the FIM spectrum concentrates in a small number of directions in the tested architectures, and its top eigenvectors are associated with low-frequency output structures. This empirical finding has not been thoroughly reported.
- PhysGuard improves most evaluation metrics across four neural operator architectures and three scenarios from RealPDEBench. To our knowledge, this is the first systematic study of knowledge-preserving fine-tuning for neural PDE surrogates.

2 Related Work

Neural operators for PDE surrogate modelling. Conventional PDE solvers are often too expensive for tasks that require repeated evaluation. Neural operators address this challenge by learning mappings between function spaces, such as from an initial condition or coefficient field to a PDE solution, and serving as fast surrogates [3]. Concretely, a neural operator maps a discretized input function (e.g., an initial condition or a forcing field) to a discretized output function. Representative architectures include FNO [16] and its factorised variant [19], CNO [20], DeepONet [17], Transolver [18], and DPOT [21]. Recent efforts have also focused on multi-spatiotemporal-scale generalisation [22]. Detailed descriptions of the operators used in this work are given in Appendix C.1. However, many existing models are trained and benchmarked only on simulation data [16, 17, 3],

and current benchmarks such as PDEBench [23] and CFDBench [24] provide only simulated ground truth. As a result, deploying these models on real-world measurements affected by sensor noise and systematic domain shift remains largely unexplored. RealPDEBench [6] provides the first benchmark with paired numerical and experimental data for this problem.

Sim-to-real transfer and physics-informed learning. Bridging the simulation-to-reality gap has been widely studied in robotics and computer vision, where domain randomisation [25, 26] and feature alignment [27] are standard tools. A complementary line of research introduces physical priors directly into the learning process. Physics-informed neural networks [28] enforce PDE residual constraints, while physics-informed machine learning more broadly incorporates structural knowledge into model architectures and training objectives [1]. However, these methods primarily aim to improve generalisation during training or rely on physics supervision throughout optimisation. They do not directly address the challenge of adapting an already-trained surrogate to a new domain while preserving the physics it has already acquired. Appendix B discusses several potential approaches to transfer learning for neural operators.

Knowledge-preserving adaptation. Catastrophic forgetting [29] is a central challenge when adapting pre-trained models to new domains. In the broader machine learning community, a rich set of strategies has been developed. Regularisation-based methods such as EWC [13], Synaptic Intelligence [30], and L_2-SP [12] penalise deviations from pre-trained weights. Parameter-efficient methods such as LoRA [15] and Adapters [31] freeze the backbone and insert small trainable modules. GPM [14] projects gradients away from activation subspaces, whereas GEM [32] constrains interference with stored examples. OGD [33] and ORTHOG-SUBSPACE [34] also use orthogonality constraints. Similar ideas have also been explored in the context of knowledge editing in language models, as exemplified by AlphaEdit [35] and CrispEdit [36]. PhysGuard shares the gradient-projection mechanism with prior work but differs in its subspace construction. GPM builds its subspace from intermediate representations accumulated across a sequence of tasks, whereas PhysGuard constructs the subspace from loss-level Fisher information on a single simulation dataset, and targets the preservation of low-frequency physics rather than sequential task knowledge. These techniques have been extensively studied for language and vision models, yet their application to neural operator fine-tuning remains less explored. Appendix B.2 compares these methods and their applicability.

3 Method: Fisher-Guided Gradient Projection

In this section, we present our approach to sim-to-real adaptation for neural operators. We first formulate the adaptation problem (Section 3.1), then identify physics-critical parameter directions via the empirical Fisher Information Matrix [37, 38] using a Gram-matrix formulation to address the associated computational challenges (Section 3.2), and finally constrain fine-tuning gradients to the complementary safe subspace (Section 3.3).

3.1 Problem Formulation

Consider a neural operator model $f_\theta(\cdot)$ pretrained on a large simulation dataset $\mathcal{D}_{\text{sim}} = \{(x_i, y_i)\}_{i=1}^{M}$, yielding trainable parameters $\theta^* \in \mathbb{R}^d$ that capture the underlying physics of PDEs. Our goal is to adapt this model to real experimental data $\mathcal{D}_{\text{real}} = \{(\tilde{x}_j, \tilde{y}_j)\}_{j=1}^{M'}$ during fine-tuning, while retaining the physical knowledge encoded in pretraining.

As shown in Figure 1, the core physics learned from simulation is reflected in the low-frequency components of the solution, which encode large-scale, globally coherent structures. We therefore use a low-frequency reconstruction loss $\mathcal{L}_{\text{low-f}}$ to quantify physics preservation, and express the desired fine-tuning objective as:

$$\min_{\theta} \underbrace{\mathcal{L}_{\text{real}}(\theta) = \frac{1}{M'} \sum_{j=1}^{M'} \ell\big(f_\theta(\tilde{x}_j),\, \tilde{y}_j\big)}_{\text{adapt to real data}} \quad \text{s.t.} \quad \underbrace{\mathcal{L}_{\text{low-f}}(\theta) \le \mathcal{L}_{\text{low-f}}(\theta^*)}_{\text{preserve pre-trained knowledge}}, \tag{1}$$

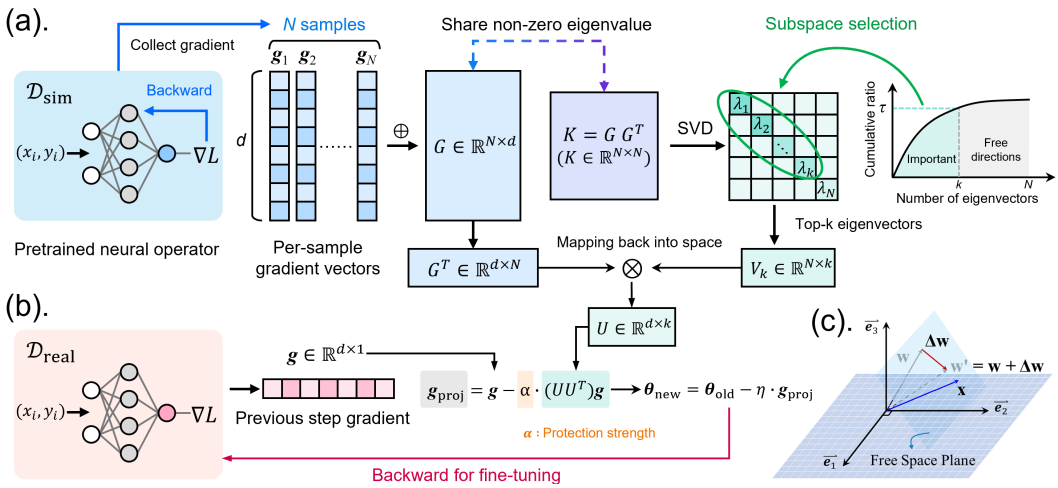


Figure 2: Overview of the PhysGuard framework. (a) Gradient vectors from a pre-trained model are stacked into $\boldsymbol{G}$, and SVD of the Gram matrix $\boldsymbol{K}$ is applied to identify the physics-critical subspace $\boldsymbol{U}$; (b) During fine-tuning, each gradient $\boldsymbol{g}$ is projected onto the complement of $\boldsymbol{U}$, ensuring parameter updates do not overwrite physical knowledge learned during pre-training; (c) Visualization of the null-space projection. Best viewed in color.

where $\mathcal{L}_{\text{real}}$ is the loss in real data, and $\mathcal{L}_{\text{low-f}}$ measures the prediction error on the low-frequency components of the solution.

Immediately enforcing this constraint with a penalty term can introduce additional hyperparameters [13, 12, 30]. Instead, we adopt a geometric perspective whereby fine-tuning updates are restricted to parameter directions along which the pretrained loss is insensitive, approximately satisfying the constraint by construction. The key question then becomes how to identify these directions, which we will address next.

3.2 Physics Subspace Identification

In Section 3.1, we define the sim-to-real training objective for neural operators, where the key challenge focuses on identifying the physics-critical subspaces from the pretrained model. This section describes how we achieve this in three steps.

Fisher Information Matrix. Since neural operator architectures vary widely, we require an architecture-agnostic measure of prediction sensitivity with respect to each parameter direction. The empirical Fisher Information Matrix (FIM) serves this purpose. Given N samples from the simulation dataset, we perform a forward and backward pass with θ^* held fixed to evaluate the per-sample gradient $\boldsymbol{g}_i = \nabla_\theta \ell(\theta^*; x_i, y_i) \in \mathbb{R}^d$, where d is the dimension of parameters. Stacking these into a matrix $\boldsymbol{G} = [\boldsymbol{g}_1, \ldots, \boldsymbol{g}_N]^\top \in \mathbb{R}^{N \times d}$, the empirical FIM is then:

$$\boldsymbol{F} = \frac{1}{N} \boldsymbol{G}^\top \boldsymbol{G} \in \mathbb{R}^{d \times d}. \tag{2}$$

Eigenvectors of $\boldsymbol{F}$ with large eigenvalues identify parameter directions along which predictions are most sensitive, corresponding to the physics-critical directions we wish to protect. Conversely, eigenvectors with small or zero eigenvalues span directions along which predictions are largely insensitive, representing directions available for adaptation.

Intuitively, because the pretrained model is optimized on smooth simulation data that is dominated by large-scale physical patterns, the directions to which the loss is most sensitive may be associated with output modes encoding these low-frequency structures. We empirically verify this connection in Section 4.2.

Efficient eigendecomposition based on a Gram matrix. Since the parameter dimension d can reach millions, directly eigendecomposing the FIM poses a significant computational and memory

challenge. To overcome this, we exploit the identity that $\boldsymbol{G}^\top\boldsymbol{G}$ and $\boldsymbol{G}\boldsymbol{G}^\top$ share the same non-zero eigenvalues. We therefore construct the much smaller Gram matrix[1] $\boldsymbol{K} = \boldsymbol{G}\boldsymbol{G}^\top \in \mathbb{R}^{N\times N}$ and compute its decomposition as follows.

$$\left\{\boldsymbol{V},\ \boldsymbol{\Lambda},\ \boldsymbol{V}^\top\right\} = \mathrm{SVD}(\boldsymbol{K}) \tag{3}$$

where $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1, \ldots, \lambda_N)$ with $\lambda_1 \geq \cdots \geq \lambda_N \geq 0$, and the columns of $\boldsymbol{V} \in \mathbb{R}^{N\times N}$ are the corresponding eigenvectors. Because $\boldsymbol{K}$ is symmetric positive semi-definite, this is an eigendecomposition (equivalently, an SVD with matching left and right singular vectors). The corresponding FIM eigenvectors in parameter space are given by $u_j = \boldsymbol{G}^\top v_j/\sqrt{\lambda_j}$ for unit v_j and $\lambda_j > 0$, with Fisher eigenvalues λ_j/N, thereby avoiding the explicit construction of the $d \times d$ matrix. We provide a detailed proof of the eigenvalue correspondence and the recovery procedure in Appendix D.

Adaptive subspace selection. The SVD provides a ranked list of eigenvectors, but the number of directions to protect is not fixed. To select the most relevant ones adaptively, we define the cumulative fraction of Fisher information captured by the top k eigenvalues:

$$\rho(k) = \frac{\sum_{j=1}^{k}\lambda_j}{\sum_{j=1}^{N}\lambda_j}. \tag{4}$$

We choose the smallest k such that $\rho(k) \geq \tau$, and then assemble the physics-subspace basis $\boldsymbol{V}_k = [v_1, \ldots, v_k] \in \mathbb{R}^{N\times k}$ from the corresponding top-k eigenvectors. [2]

Subsequently, these directions are mapped back to the original parameter space via $\boldsymbol{U} = \mathrm{normalise}(\boldsymbol{G}^\top\boldsymbol{V}_k) \in \mathbb{R}^{d\times k}$, where each column is normalized. The columns of $\boldsymbol{U}$ span the critical physics subspace, while its orthogonal complement defines the safe subspace for fine-tuning. A detailed illustration is provided in Figure 2(a).

3.3 Constrained Fine-Tuning via Orthogonal Projection

Having identified the physics-critical subspace from Section 3.2, we now describe how to exploit it during fine-tuning. The key idea is to constrain each gradient update to lie in the orthogonal complement of this subspace, so that adaptation to real data cannot overwrite the physics structure encoded by pretraining. Concretely, given a gradient $\boldsymbol{g} \in \mathbb{R}^d$ and the subspace basis $\boldsymbol{U}$ for a parameter group, we project out its physics-critical component:

$$\boldsymbol{g}_{\mathrm{proj}} = \boldsymbol{g} - \alpha \cdot \boldsymbol{U}\boldsymbol{U}^\top\boldsymbol{g} \tag{5}$$

where $\alpha \in [0, 1]$ controls the strength of protection. $\alpha = 0$ recovers standard fine-tuning, while $\alpha = 1$ enforces full projection onto the null space of $\boldsymbol{U}^\top$, removing updates along physics-critical directions. Values in between provide a softer trade-off, partially suppressing these directions while still allowing some movement along them.

In practice, each layer m maintains its own subspace basis $\boldsymbol{U}^{(m)}$, estimated independently from simulation data as described in Section 3.2. The projection in Equation (5) is then applied per-layer after every backward pass. Figure 2(b) illustrates this procedure. The complete two-phase workflow (offline subspace estimation followed by iterative constrained fine-tuning) is formalized in Algorithm 1. Crucially, the subspace is computed only once and can be reused across multiple downstream tasks without recomputation. Further analysis of the projected updates is provided in Appendix I.

In our setting, spectral weights in architectures such as FNO are parameterized as complex numbers. To ensure compatibility with standard operations, we decompose each complex-valued gradient into its real and imaginary components. These are then used to construct the Fisher subspace and perform the projection. Further details are provided in Appendix E.

4 Experiments

In this section, we conduct experiments to address the following research questions:

[1] A small subset of samples suffices to estimate the subspace in the tested configurations. See Appendix D for details.

[2] The same threshold is used in all main experiments, please see Appendix G.3 for a detailed discussion.

Algorithm 1 PhysGuard: Fisher-Guided Gradient Projection for Sim-to-Real Adaptation

Require: Pre-trained θ^*, simulation data $\mathcal{D}_{\text{sim}}$, real data $\mathcal{D}_{\text{real}}$, samples N, threshold τ, strength α

Phase 1: Subspace Estimation (one-time)
for each layer $m = 1, \ldots, L$ **do**
  for $i = 1, \ldots, N$ **do**
    Sample $(x_i, y_i) \sim \mathcal{D}_{\text{sim}}$; compute $\boldsymbol{g}_i^{(m)} = \nabla_{\theta^{(m)}} \ell(\theta^*; x_i, y_i)$
  end for
  $\boldsymbol{G}^{(m)} \leftarrow [\boldsymbol{g}_1^{(m)}, \ldots, \boldsymbol{g}_N^{(m)}]^\top$; $\boldsymbol{K}^{(m)} \leftarrow \boldsymbol{G}^{(m)} \boldsymbol{G}^{(m)\top}$
  $\{\boldsymbol{V}^{(m)}, \boldsymbol{\Lambda}^{(m)}\} \leftarrow \text{SVD}(\boldsymbol{K}^{(m)})$; select k_m via Equation (4)
  $\boldsymbol{V}_{k_m}^{(m)} \leftarrow [\boldsymbol{v}_1^{(m)}, \ldots, \boldsymbol{v}_{k_m}^{(m)}]$; $\boldsymbol{U}^{(m)} \leftarrow \text{normalise}\left(\boldsymbol{G}^{(m)\top} \boldsymbol{V}_{k_m}^{(m)}\right)$
end for
Phase 2: Constrained Fine-Tuning (iterative)
for each step $t = 1, \ldots, T$ **do**
  Sample batch $\mathcal{B} \sim \mathcal{D}_{\text{real}}$; forward & backward to obtain $\boldsymbol{g}^{(m)}$ for all m
  for each layer m **do**
    $\boldsymbol{g}_{\text{proj}}^{(m)} \leftarrow \boldsymbol{g}^{(m)} - \alpha \boldsymbol{U}^{(m)} \left(\boldsymbol{U}^{(m)\top} \boldsymbol{g}^{(m)}\right)$ ▷ *Equation* (5)
  end for
  Update θ with optimiser using $\{\boldsymbol{g}_{\text{proj}}^{(m)}\}$
end for
return Adapted parameters θ_{adapted}

- **RQ1**: Do the FIM eigenvectors align with low-frequency physics?
- **RQ2**: Can PhysGuard improve sim-to-real performance across architectures and scenarios?
- **RQ3**: Can PhysGuard preserve low-frequency physical structures during real-data adaptation?

4.1 Experimental Setup

Benchmark. We adopt RealPDEBench [6], which pairs real experimental measurements with matched numerical simulations. We select three scenarios of increasing difficulty: **cylinder flow**, **controlled cylinder**, and **turbulent combustion**. Details are provided in Appendix C.3.

Architectures. We evaluate four neural operator architectures (3.5M to 50.4M parameters): **FNO** [16] (Fourier-domain convolution), **CNO** [20] (multi-resolution CNN), **DeepONet** [17] (branch-trunk factorization), and **Transolver** [18] (physics-aware attention). Configurations are in Appendix C.3.

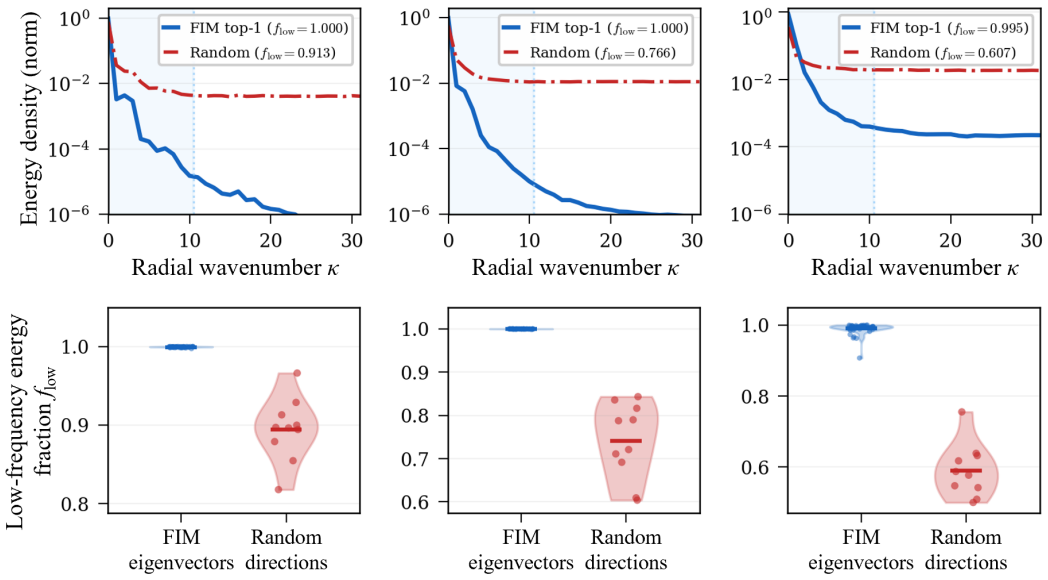
Methods. We compare five approaches. **Pretrained** applies the simulation-trained model directly without adaptation. **DFT** (Direct Fine-Tuning) updates all parameters on real data without constraints. **L_2-SP** [12] penalizes deviation from pretrained weights. **EWC** [13] applies a diagonal Fisher penalty to protect important parameters. **PhysGuard** projects gradients away from the physics-critical subspace ($\alpha = 1.0$). Additional comparisons with LoRA, layer freezing, and matched-rank random projection, along with ablations on α and τ, are reported in Appendix G.

Training. All methods share one fine-tuning budget with checkpoint selection based on the lowest validation RMSE. We report test-set metrics at the selected checkpoint, keeping the comparison focused on the update rule rather than per-method tuning. Protocol details and multi-seed results are in Appendices C.3 and H.

Metrics. We report both data-oriented metrics (**RMSE**, R^2) and physics-oriented metrics (**fRMSE**, its **Low-f** band, and **relative low-frequency error**). Full metric definitions and additional results are provided in Appendix C.2.

4.2 FIM Eigenvectors and Low-Frequency Physics (RQ1)

While the low intrinsic dimensionality of neural network parameter spaces has been established for language models and image classifiers [39, 40], it remains unexamined for neural operators trained on physical simulations. We therefore first verify this assumption before evaluating transfer performance, testing whether leading Fisher directions affect low-frequency output structures.



(a). Cylinder Flow (b). Control Cylinder (c). Turbulent Combustion

Figure 3: Diagnostic spectral probe for FIM eigenvectors. *Top*: Radial energy spectrum of the output perturbation induced by a FIM top-1 eigenvector vs. a random direction; shaded region = low-frequency band. *Bottom*: Low-frequency energy fraction f_{low} for the probed leading FIM eigenvectors vs. random directions. Each point is one direction. See Appendix F for detailed definitions. Best viewed in color.

We design a spectral probe to test this on a pretrained FNO (full protocol in Appendix F). For a given parameter direction v, we slightly perturb the pretrained weights along v and observe how the model output changes. We then apply a 2D Fourier transform to this output change and summarize it by the radial wavenumber κ, which characterizes spatial scale. Small κ corresponds to large-scale patterns such as vortices, whereas large κ captures fine-grained fluctuations. We define f_{low} as the fraction of total spectral energy in the low-frequency band, namely the lowest third of κ modes. When $f_{\text{low}} \approx 1.0$, the direction mainly affects large-scale structures. When it is small, the effect is distributed across higher frequencies.

We compare the leading Fisher directions from the diagnostic probe described in Appendix F with random directions as a control. Figure 3 (top) compares the FIM top-1 eigenvector with a random direction. Across all three scenarios, the FIM eigenvector produces perturbations concentrated almost entirely at low frequencies ($f_{\text{low}} \geq 0.995$), while random directions spread energy more uniformly. Figure 3 (bottom) extends this to all probed Fisher directions, confirming a consistent trend.

These results support PhysGuard's design by linking sensitive parameter directions in the pretrained model to low-frequency output structures. A similar pattern is observed in probes of CNO, DeepONet, and Transolver (Appendix F.4).

4.3 Sim-to-Real Transfer Performance (RQ2)

Table 1 reports representative metrics across all architectures and scenarios. We summarize the main observations below.

Consistent improvement across architectures. PhysGuard achieves competitive performance in most metrics in Table 1. All exceptions occur in Controlled Cylinder, where the sim-to-real gap is small, and all methods perform similarly. Multi-seed comparisons show that PhysGuard's gains over DFT on Cylinder Flow consistently exceed the standard deviation across seeds. On Controlled Cylinder and Turbulent Combustion, however, several smaller margins fall within one standard deviation and cannot be distinguished from seed variation (Appendix H.1).

Table 1: Statistical results across all RealPDEBench scenarios. The best and second-best results are highlighted in **Blue** and Green. Best viewed in color.

Architecture	Method	Cylinder Flow				Controlled Cylinder				Turbulent Combustion			
		RMSE ↓	R^2 ↑	fRMSE ↓	Low-f ↓	RMSE ↓	R^2 ↑	fRMSE ↓	Low-f ↓	RMSE ↓	R^2 ↑	fRMSE ↓	Low-f ↓
FNO (50.4 M)	Pretrained	0.08060	0.6530	0.01266	0.01829	0.02788	0.8455	0.00566	0.00646	0.03626	0.2735	0.00507	0.00811
	DFT	0.06487	0.7753	0.01048	0.01651	**0.00997**	0.9802	**0.00119**	**0.00106**	0.02548	0.6414	0.00334	0.00419
	L_2-SP	0.06556	0.7704	0.01075	0.01739	0.01191	0.9718	0.00139	0.00126	0.02779	0.5732	0.00374	0.00494
	EWC	0.06316	0.7869	0.01031	0.01596	**0.00997**	**0.9803**	0.00120	0.00108	0.02544	0.6424	0.00333	0.00417
	PhysGuard	**0.06036**	**0.8054**	**0.00914**	**0.01131**	**0.00997**	0.9802	0.00121	0.00110	**0.02532**	**0.6458**	**0.00331**	**0.00414**
CNO (8.0 M)	Pretrained	0.07239	0.7201	0.01108	0.01379	0.01652	0.9458	0.00226	0.00181	0.04265	−0.0049	0.00603	0.01057
	DFT	0.04988	0.8671	0.00865	0.01585	0.00885	0.9844	0.00105	0.00091	0.02688	0.6008	0.00361	0.00483
	L_2-SP	0.04915	0.8710	0.00855	0.01506	0.00936	0.9826	0.00113	0.00100	0.02685	0.6018	0.00360	0.00481
	EWC	0.04855	0.8741	0.00773	0.01355	0.00886	0.9844	0.00105	0.00091	0.02670	0.6060	0.00358	0.00476
	PhysGuard	**0.04635**	**0.8853**	**0.00759**	**0.01148**	**0.00884**	**0.9845**	**0.00104**	**0.00088**	**0.02648**	**0.6125**	**0.00352**	**0.00462**
DeepONet (3.5 M)	Pretrained	0.10072	0.4581	0.01612	0.02793	0.05065	0.4903	0.01239	0.01284	0.04036	0.0999	0.00562	0.00947
	DFT	0.09309	0.5371	0.01507	0.02527	**0.03004**	**0.8207**	**0.00570**	0.00446	0.02757	0.5799	0.00369	0.00448
	L_2-SP	0.09121	0.5556	0.01428	0.02627	0.03223	0.7936	0.00680	0.00653	0.02805	0.5653	0.00377	0.00464
	EWC	0.09055	0.5621	0.01424	0.02599	0.03006	0.8205	**0.00570**	**0.00439**	0.02745	0.5837	0.00366	0.00444
	PhysGuard	**0.07435**	**0.7048**	**0.01246**	**0.01957**	0.03023	0.8184	0.00573	**0.00439**	**0.02743**	**0.5844**	**0.00365**	**0.00442**
Transolver (4.3 M)	Pretrained	0.11147	0.3364	0.01692	0.03362	0.02642	0.8613	0.00570	0.00703	0.04441	−0.0898	0.00623	0.01125
	DFT	0.11236	0.3257	0.01640	0.03349	0.01898	0.9284	0.00268	0.00252	0.03995	0.1181	0.00534	0.00833
	L_2-SP	0.11361	0.3107	0.01651	0.03370	0.01920	0.9268	0.00276	0.00265	0.04087	0.0773	0.00541	0.00864
	EWC	0.11330	0.3144	0.01648	0.03361	0.01898	0.9284	0.00266	0.00246	0.04070	0.0845	0.00538	0.00854
	PhysGuard	**0.10695**	**0.3891**	**0.01567**	**0.03097**	**0.01852**	**0.9318**	**0.00254**	**0.00235**	**0.03841**	**0.1848**	**0.00512**	**0.00758**

The improvement holds across architectures with fundamentally different inductive biases, ranging from FNO's spectral convolution to DeepONet's branch-trunk factorisation and Transolver's physics-attention, suggesting that the FIM-based subspace captures properties of the learned physics shared by these architectures. Among the baselines, L_2-SP and EWC are less stable. L_2-SP underperforms DFT on several FNO metrics in the Controlled Cylinder scenario, indicating that a uniform ℓ_2 penalty can over-restrict adaptation along directions that are not physics-critical. EWC performs close to DFT but rarely surpasses it by a large margin, consistent with its diagonal Fisher approximation not capturing the correlations among parameter directions that the full FIM eigenvectors encode. PhysGuard, by projecting gradients away from the physics-critical subspace rather than penalising individual weights, selectively blocks only the most sensitive directions while leaving the rest unconstrained.

DFT can degrade low-frequency structures. On Cylinder Flow, DFT raises the Low-f error above the pretrained baseline for CNO. This means standard fine-tuning can overwrite large-scale physical patterns even when overall RMSE improves. PhysGuard avoids this regression and reduces Low-f by 22% to 32% relative to DFT on FNO, CNO, and DeepONet.

Larger gains under larger domain shifts. PhysGuard's gains over DFT are most evident on Cylinder Flow, where pretrained R^2 values range from 0.34 to 0.72 and PhysGuard delivers the largest improvements. For DeepONet, PhysGuard achieves a relative R^2 improvement of over 50%, compared to less than 20% for DFT. On Transolver, all three baselines slightly degrade Cylinder Flow RMSE and R^2, with DFT even increasing RMSE, while PhysGuard still yields a relative R^2 gain of approximately 16%.

This pattern is consistent with reduced interference along protected directions, although the metrics alone do not establish the gradient geometry. Matched-rank random controls further support Fisher-based direction selection (Appendix I). On Controlled Cylinder, inter-method differences are small. On Turbulent Combustion, an intermediate-shift scenario, PhysGuard remains effective but with narrower margins.

4.4 Low-Frequency Preservation (RQ3)

Qualitative comparison. Figure 4 compares predicted flow fields. In panel (a), the ground truth shows alternating vortex cores (Kármán vortex street). The pretrained DeepONet produces a nearly uniform field. DFT and EWC recover some structure but miss the sharp vortex boundaries. PhysGuard produces the closest match to ground truth, consistent with its lower Low-f error in Table 1. In panel (b), all methods track the ground truth well because the sim-to-real gap is small on this scenario.

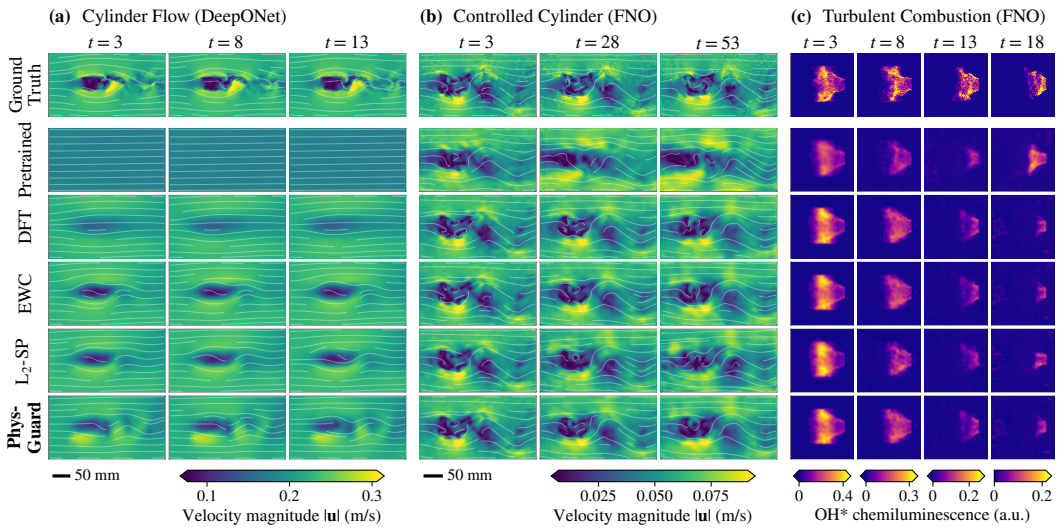


Figure 4: Predicted fields on three scenarios. (a) Cylinder Flow (DeepONet), (b) Controlled Cylinder (FNO), (c) Turbulent Combustion (FNO). Each panel shows Ground Truth, Pretrained, DFT, EWC, L_2-SP, and PhysGuard at selected time steps. Full per-architecture results are in Appendix J. Best viewed in color.

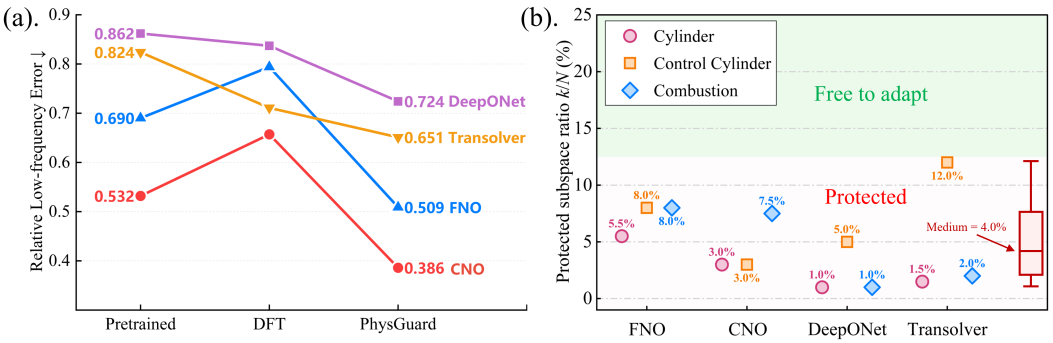


Figure 5: (a) Relative low-frequency error (↓) on Cylinder Flow for the pretrained model, DFT, and PhysGuard. (b) Protected subspace ratio k/N (%) per architecture and scenario. Here N is the number of sampled gradients and the green shading is schematic. Best viewed in color.

In panel (c), the combustion field has localized high-intensity zones. Pretrained predictions are blurred. All baselines improve substantially, with PhysGuard producing slightly sharper boundaries.

Low-frequency error across architectures. Figure 5(a) compares relative low-frequency error on Cylinder Flow. This metric normalizes low-frequency spectral errors by the corresponding reference amplitudes, providing a dimensionless measure of large-scale flow fidelity that complements overall accuracy metrics (Appendix C.2). PhysGuard reduces this error relative to the pretrained model by 26.1% (FNO), 27.4% (CNO), 21.0% (Transolver), and 16.0% (DeepONet). Table H.8 reports results for all three scenarios. Figure 5(b) shows the protected subspace ratio k/N. Across all 12 architecture-scenario combinations, this ratio ranges from 1.0% to 12.0%, with a median of 4.0%. Because N bounds the rank of the estimated FIM, k/N is the fraction of candidate directions that PhysGuard protects. As a fraction of the full parameter space, the protected subspace is much smaller (k/d, since $N \ll d$).

5 Limitations

PhysGuard is designed for settings where the FIM spectrum exhibits a low-rank structure, as a compact subspace can then efficiently capture the directions most critical to physical knowledge. This structure is observed across the four architectures in our main experiments, including FNO, CNO, DeepONet, and Transolver. We note that PhysGuard may be less effective for certain pretrained operators (e.g., DPOT [21]) whose Fisher spectrum does not exhibit a clear low-rank pattern. We report this result in Appendix K. The four-stage analysis there shows spectrum flattening during scenario-specific training, which cannot be attributed solely to joint pretraining or the backbone. The leading Fisher directions must also correspond to stable signal structures; whether this holds in other signal domains remains untested.

Beyond this dependence on the estimated spectrum, our evaluation is limited to fluid-mechanics scenarios from RealPDEBench [6]. To our knowledge, this is currently the only benchmark in the community that provides results on real experimental data, which is why we focus our evaluation here. That said, whether PhysGuard generalises to other PDE families and sensing modalities remains an open question, and we hope this work encourages the SciML community to develop broader real-world benchmarks for evaluating neural PDE solvers.

6 Conclusion

We presented PhysGuard, a framework that preserves pretrained physical knowledge during sim-to-real adaptation of neural PDE surrogates through Fisher-guided gradient projection. The key idea is to project fine-tuning gradients onto the orthogonal complement of the physics-critical subspace identified by the empirical Fisher Information Matrix, which reduces low-frequency degradation in the tested settings without requiring additional penalty terms. Experiments on RealPDEBench across four architectures and three physical scenarios confirm consistent improvements, particularly in low-frequency fidelity and under large domain shifts. That said, challenges remain in scaling to foundation-model-scale operators and updating the protected subspace during adaptation, both of which we see as promising directions for future work.

We have recently noted that graph neural simulators offer another promising approach to modelling complex physical systems [41–43]. An interesting extension of this work would be to explore how these simulators can retain useful prior physical knowledge while learning new dynamics across different physical scenarios.

Acknowledgments and Disclosure of Funding

The first author acknowledges financial support from the Melbourne Research Scholarship (MRS) awarded by the University of Melbourne. This work was also supported in part by the Australian Department of Education under grant AEA IV240100185.

References

[1] George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

[2] Steven L Brunton, Bernd R Noack, and Petros Koumoutsakos. Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics*, 52:477–508, 2020.

[3] 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.

[4] Michael McCabe, Bruno Régaldo-Saint Blancard, Liam Parker, Ruben Ohana, Miles Cranmer, Alberto Bietti, Michael Eickenberg, Siavash Golkar, Geraud Krawezik, François Lanusse, et al. Multiple physics pretraining for spatiotemporal surrogate models. In *Advances in Neural Information Processing Systems*, 2024.

[5] Maximilian Herde, Bogdan Raonić, Tobias Rohner, Roger Käppeli, Roberto Molinaro, Emmanuel de Bézenac, and Siddhartha Mishra. Poseidon: Efficient foundation models for PDEs. In *Advances in Neural Information Processing Systems*, 2024.

[6] Peiyan Hu, Haodong Feng, Hongyuan Liu, Tongtong Yan, Wenhao Deng, Tianrun Gao, Rong Zheng, Haoren Zheng, Chenglei Yu, Chuanrui Wang, Kaiwen Li, Zhi-Ming Ma, Dezhi Zhou, Xingcai Lu, Dixia Fan, and Tailin Wu. RealPDEBench: A benchmark for complex physical systems with real-world data. In *International Conference on Learning Representations*, 2026.

[7] Arvind Mohan, Ashesh Chattopadhyay, and Jonah Miller. What you see is not what you get: Neural partial differential equations and the illusion of learning. *arXiv preprint arXiv:2411.15101*, 2024.

[8] Jiayun Wang, Oleksii Ostras, Masashi Sode, Bahareh Tolooshams, Zongyi Li, Kamyar Azizzadenesheli, Gianmarco Pinton, and Anima Anandkumar. Ultrasound lung aeration map via physics-aware neural operators. *arXiv preprint arXiv:2501.01157*, 2025.

[9] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *International Conference on Machine Learning*, 2019.

[10] Zhi-Qin John Xu, Yaoyu Zhang, and Yanyang Xiao. Training behavior of deep neural network in frequency domain. In *International Conference on Neural Information Processing*, 2019.

[11] Shaoxiang Qin, Fuyuan Lyu, Wenhui Peng, Dingyang Geng, Ju Wang, Xing Tang, Sylvie Leroyer, Naiping Gao, Xue Liu, and Liangzhu Leon Wang. Toward a better understanding of Fourier neural operators from a spectral perspective. *arXiv preprint arXiv:2404.07200*, 2024.

[12] Xuhong Li, Yves Grandvalet, and Franck Davoine. Explicit inductive bias for transfer learning with convolutional networks. In *International Conference on Machine Learning*, pages 2825–2834, 2018.

[13] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

[14] Gobinda Saha, Isha Garg, and Kaushik Roy. Gradient projection memory for continual learning. In *International Conference on Learning Representations*, 2021.

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

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

[17] 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.

[18] 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.

[19] Alasdair Tran, Alexander Mathews, Lexing Xie, and Cheng Soon Ong. Factorized Fourier neural operators. In *International Conference on Learning Representations*, 2023.

[20] Bogdan Raonić, Roberto Molinaro, Tim De Ryck, Tobias Rohner, Francesca Bartolucci, Rima Alaifari, Siddhartha Mishra, and Emmanuel de Bézenac. Convolutional neural operators for robust and accurate learning of PDEs. In *Advances in Neural Information Processing Systems*, 2023.

[21] Zhongkai Hao, Chang Su, Songming Liu, Julius Berner, Chengyang Ying, Hang Su, Anima Anandkumar, Jian Song, and Jun Zhu. DPOT: Auto-regressive denoising operator transformer for large-scale PDE pre-training. In *International Conference on Machine Learning*, 2024.

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

[23] Makoto Takamoto, Timothy Praditia, Raphael Leiteritz, Daniel MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. PDEBench: An extensive benchmark for scientific machine learning. In *Advances in Neural Information Processing Systems*, 2022.

[24] Yining Luo, Yingfa Chen, and Zhen Zhang. CFDBench: A large-scale benchmark for machine learning methods in fluid dynamics. *arXiv preprint arXiv:2310.05963v2*, 2024.

[25] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017.

[26] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *IEEE International Conference on Robotics and Automation*, 2018.

[27] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *Journal of Machine Learning Research*, 17(59):1–35, 2016.

[28] Maziar Raissi, Paris Perdikaris, and George Em 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.

[29] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of Learning and Motivation*, volume 24, pages 109–165. Elsevier, 1989.

[30] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, 2017.

[31] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning*, pages 2790–2799, 2019.

[32] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, 2017.

[33] Mehrdad Farajtabar, Navid Azizan, Alex Mott, and Ang Li. Orthogonal gradient descent for continual learning. In *International Conference on Artificial Intelligence and Statistics*, pages 3762–3773, 2020.

[34] Arslan Chaudhry, Naeemullah Khan, Puneet K Dokania, and Philip H S Torr. Continual learning in low-rank orthogonal subspaces. In *Advances in Neural Information Processing Systems*, 2020.

[35] Junfeng Fang, Houcheng Jiang, Kun Wang, Yunshan Ma, Jie Shi, Xiang Wang, Xiangnan He, and Tat-Seng Chua. AlphaEdit: Null-space constrained knowledge editing for language models. In *International Conference on Learning Representations*, 2025.

[36] Zarif Ikram, Arad Firouzkouhi, Stephen Tu, Mahdi Soltanolkotabi, and Paria Rashidinejad. CrispEdit: Low-curvature projections for scalable non-destructive LLM editing. *arXiv preprint arXiv:2602.15823*, 2026.

[37] Shun-ichi Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2): 251–276, 1998.

[38] James Martens. New insights and perspectives on the natural gradient method. *Journal of Machine Learning Research*, 21(146):1–76, 2020.

[39] Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. Measuring the intrinsic dimension of objective landscapes. In *International Conference on Learning Representations*, 2018.

[40] Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 7319–7328. Association for Computational Linguistics, 2021.

[41] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 8459–8468. PMLR, 2020.

[42] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W. Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2021.

[43] Changjian Zhou, Negin Yousefpour, Jie Qi, Junfeng Fang, Guillermo A. Narsilio, and Hans Petter Jostad. TRACE: A spatiotemporal contact memory graph network simulator for granular dynamics. *arXiv preprint arXiv:2609.02991*, 2026.

[44] Somdatta Goswami, Katiana Kontolati, Michael D Shields, and George Em Karniadakis. Deep transfer operator learning for partial differential equations under conditional shift. *Nature Machine Intelligence*, 4(12):1155–1164, 2022.

[45] Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. Physics-informed neural operator for learning partial differential equations. *ACM/IMS Journal of Data Science*, 1(3):1–27, 2024.

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

[47] Hangwei Zhang, Chun Kang, Yan Wang, and Difan Zou. F-Adapter: Frequency-adaptive parameter-efficient fine-tuning in scientific machine learning. In *Advances in Neural Information Processing Systems*, 2025.

[48] Peng Fan and Guofei Pang. Convolutional-neural-operator-based transfer learning for solving PDEs. *arXiv preprint arXiv:2512.17969*, 2025.

[49] Md Ashiqur Rahman, Robert Joseph George, Mogab Elleithy, Daniel Leibovici, Zongyi Li, Boris Bonev, Colin White, Julius Berner, Raymond A Yeh, Jean Kossaifi, Kamyar Azizzadenesheli, and Anima Anandkumar. Pretraining codomain attention neural operators for solving multiphysics PDEs. In *Advances in Neural Information Processing Systems*, 2024.

[50] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *European Conference on Computer Vision*, 2018.

[51] Shipeng Wang, Xiaorong Li, Jian Sun, and Zongben Xu. Training networks in null space of feature covariance for continual learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 184–193, 2021.

[52] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. iCaRL: Incremental classifier and representation learning. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017.

[53] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip H S Torr, and Marc'Aurelio Ranzato. On tiny episodic memories in continual learning. *arXiv preprint arXiv:1902.10486*, 2019.

[54] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with A-GEM. In *International Conference on Learning Representations*, 2019.

[55] Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline. In *Advances in Neural Information Processing Systems*, 2020.

[56] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[57] Arun Mallya and Svetlana Lazebnik. PackNet: Adding multiple tasks to a single network by iterative pruning. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[58] Joan Serrà, Dídac Surís, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *International Conference on Machine Learning*, pages 4548–4557, 2018.

[59] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *IEEE/CVF International Conference on Computer Vision*, pages 4195–4205, 2023.

Appendix Contents

A Broader Impacts . . . **16**

B Extended Related Work . . . **16**
B.1 Neural Operator Transfer and Adaptation . . . 16
B.2 Continual Learning Methods . . . 16
B.3 Comparison with ORTHOG-SUBSPACE . . . 17

C Implementation Details . . . **18**
C.1 Architectures . . . 18
C.2 Evaluation Metrics . . . 18
C.3 Experimental Setup . . . 20
C.4 Adaptation Methods . . . 22

D Fisher Subspace Estimation . . . **23**
D.1 Fisher Information and Loss Sensitivity . . . 23
D.2 Rank Bound . . . 23
D.3 Gram Matrix Formulation . . . 23
D.4 Subspace Recovery . . . 24
D.5 Computational Cost . . . 24
D.6 Sensitivity to Sample Size . . . 25

E Complex-Valued Weights in FNO . . . **26**
E.1 Spectral Weights . . . 26
E.2 Complex Gradients . . . 26
E.3 Real-Imaginary Concatenation . . . 26
E.4 PyTorch and Wirtinger Calculus . . . 26

F Spectral Probes of Fisher Directions . . . **27**
F.1 FNO Probe Procedure . . . 27
F.2 FNO Probe Directions . . . 27
F.3 FNO Probe Layer . . . 27
F.4 Results across Architectures . . . 27

G Additional Baselines and Ablations . . . **28**
G.1 Additional Adaptation Baselines . . . 28
G.2 Projection Strength α . . . 28
G.3 Variance Threshold τ . . . 28
G.4 Scope of the Random-Subspace Control . . . 29

H Additional Results and Experimental Controls . . . **30**
H.1 Seed Variability . . . 30
H.2 Validation-Selected and Final Checkpoints . . . 31
H.3 Effective Batch-Size Control . . . 31
H.4 Official Pretrained Checkpoint Control . . . 32
H.5 Relative Low-Frequency Error . . . 33

I Subspace Selection and Orthogonality . . . **34**
I.1 Matched-Rank Random-Subspace Control . . . 34
I.2 Output-Space Probe and Leakage . . . 34
I.3 Parameter Changes and Output Drift . . . 35
I.4 Post-hoc Orthogonality Correction . . . 36

J Qualitative Visualisations . . . **37**
J.1 Cylinder Flow . . . 37
J.2 Controlled Cylinder . . . 39
J.3 Turbulent Combustion . . . 41

K DPOT-S as a Negative Case . . . **43**
K.1 Architecture . . . 43
K.2 Pretraining Configuration . . . 43
K.3 Fisher Spectrum across Training Stages . . . 43
K.4 Quantitative Results . . . 44
K.5 Qualitative Visualisations . . . 44

A Broader Impacts

PhysGuard studies how pretrained neural PDE surrogates can be adapted to real measurements while limiting changes to simulation-sensitive directions. Reusing pretrained models may reduce the need to train a separate surrogate for each experimental setting. The present results concern three fluid-mechanics scenarios and do not establish reliability in other application domains.

Residual sim-to-real error and distribution shifts remain relevant when surrogate predictions inform engineering or environmental decisions. Such use requires physics-based validation, uncertainty assessment, and domain-expert review beyond the comparisons reported here.

B Extended Related Work

This section expands the related work by reviewing transfer learning methods for neural operators, discussing the potential of continual learning approaches for operator architectures, and comparing ORTHOG-SUBSPACE with PhysGuard.

B.1 Neural Operator Transfer and Adaptation

Table B.1 summarises eight representative transfer and adaptation methods. PhysGuard uses the same Fisher-based parameter-space construction across the four evaluated operator families.

Table B.1: Transfer and adaptation methods for neural PDE operators.

Work	Backbone	Transfer or adaptation setting	Adaptation mechanism
TL-DeepONet [44]	DeepONet	Conditional shifts in PDE dynamics or geometry	Fine-tunes task-specific layers using supervised and distribution-matching objectives
PINO [45]	FNO	Instance-wise adaptation within a PDE family	Test-time optimisation using the PDE residual
Subramanian et al. [46]	FNO	Transfer across PDEs and out-of-distribution physical parameters	Mixed-physics pretraining followed by full-model fine-tuning
F-Adapter [47]	Scaled FNO	Adaptation of pretrained large operator models	Frequency-adaptive, architecture-specific adapters
Fan and Pang [48]	CNO	Few-shot transfer across several PDE systems	Compares full fine-tuning, LoRA, and neuron linear transformation
Poseidon [5]	Multiscale operator transformer	Fluid-dynamics pretraining followed by 15 downstream PDE tasks	Large-scale pretraining followed by supervised fine-tuning
DPOT [21]	Fourier-attention operator transformer	Pretraining on more than ten PDE datasets followed by downstream adaptation	Autoregressive denoising pretraining followed by fine-tuning
CoDA-NO [49]	Codomain-attention operator	Transfer from pretrained PDE systems to few-shot multiphysics tasks	Self-supervised pretraining followed by downstream fine-tuning
PhysGuard (ours)	FNO, CNO, DeepONet, and transformer operators	Simulation-to-real adaptation across different PDE scenarios	Identifies and protects simulation-sensitive Fisher directions using the same parameter-space mechanism across all backbones

Further architectural descriptions are given in Appendix C.1.

B.2 Continual Learning Methods

Preserving simulation knowledge while adapting to real data is a form of the forgetting problem studied in continual learning. Table B.2 summarises four relevant families.

Table B.2: Continual-learning methods for sim-to-real adaptation.

Family	Representative methods	Relevance to our setting	Main limitation
Regularisation	EWC [13], SI [30], MAS [50], L_2-SP [12]	Directly applicable; EWC and L_2-SP are evaluated in Table 1	Diagonal importance misses correlated parameter directions
Projection	OGD [33], ORTHOG-SUBSPACE [34], GPM [14], Adam-NSCL [51]	Most closely related to PhysGuard	Subspaces built from task sequences, gradients, or features often need architecture-specific construction
Replay	iCaRL [52], ER [53], A-GEM [54], DER++ [55]	Simulation data could be replayed during adaptation	Adds training cost and may retain simulator biases along with useful physics
Parameter isolation	Progressive Networks [56], PackNet [57], HAT [58]	Potentially useful when one operator must serve several real domains	Needs task identities, extra capacity, or parameter partitioning

EWC and L_2-SP are directly applicable and serve as baselines. PhysGuard instead protects correlated parameter directions identified by the empirical Fisher. Replay could complement this protection at additional training cost, while parameter isolation may be useful when one operator serves several domains.

B.3 Comparison with ORTHOG-SUBSPACE

ORTHOG-SUBSPACE [34] assigns low-rank orthogonal subspaces to tasks. PhysGuard estimates its protected directions from a converged simulation model. Table B.3 compares their subspace construction and use.

Table B.3: ORTHOG-SUBSPACE versus PhysGuard.

Aspect	ORTHOG-SUBSPACE	PhysGuard
Subspace construction	Predefined before training, data-independent	Estimated from the dominant eigenvectors of the empirical Fisher
Subspace interpretation	Allocates orthogonal capacity to different tasks	Identifies directions most sensitive to the pretrained simulation behaviour
Protection target	Accuracy on previously learned tasks	Physical behaviour retained from simulation pretraining
Application setting	Constrained training across a sequence of tasks	Post hoc adaptation of a converged operator, no retraining or architecture changes

C Implementation Details

C.1 Architectures

The four neural operator architectures used in the main experiments aim to learn a mapping from an input field to an output field (e.g., from initial conditions to future states). The key difference lies in how this mapping is represented and computed.

Fourier Neural Operator (FNO) [16]. FNO performs convolution in frequency space. Each layer lifts the input to a multi-channel feature map, applies a Fast Fourier Transform (FFT), retains the lowest-frequency modes, multiplies them by learned complex weights, and maps the result back to physical space via an inverse FFT. A pointwise linear bypass is added in parallel,

$$v_{\ell+1}(\mathbf{x}) = \sigma\Big(W_\ell\, v_\ell(\mathbf{x}) + \mathcal{F}^{-1}\big[R_\ell \cdot \mathcal{F}[v_\ell]\big](\mathbf{x})\Big), \tag{C.1}$$

where R_ℓ is the learned spectral filter and W_ℓ is the pointwise map. The implicit low-frequency bias makes FNO well suited to smooth, large-scale flow phenomena. In our experiments we use 4 layers, channel width 64, and mode truncations (4, 12, 16) for the cylinder scenario and (4, 16, 16) for combustion, totalling 50.4M parameters. Because FNO's spectral weights are complex-valued (`torch.complex64`), a real-embedding trick is needed during gradient projection; see Appendix E.

Convolutional Neural Operator (CNO) [20]. CNO uses anti-aliased convolutional filters whose aliasing error provably vanishes as the grid is refined, guaranteeing convergence in the L^2 norm under resolution refinement. Architecturally, CNO follows a U-Net-style hierarchy: the input is progressively downsampled (by 2× at each stage) through anti-aliased convolutions, then symmetrically upsampled, with skip connections linking matching resolution levels. Spectral normalisation and BatchNorm are applied at each resolution to promote stable Lipschitz behaviour across scales. We use 3 encoder/decoder levels, a channel multiplier of 32, and BatchNorm, giving 8.0M parameters.

Deep Operator Network (DeepONet) [17]. DeepONet builds on the universal approximation theorem for operators and trains two separate networks in parallel: a branch net that encodes the input function sampled at fixed sensor locations, and a trunk net that encodes the query location at which the output is to be evaluated. Their outputs are combined via a dot product:

$$\mathcal{G}^\dagger(u)(\mathbf{y}) \approx \sum_{k=1}^{p} b_k(u)\, t_k(\mathbf{y}) + \text{bias}, \tag{C.2}$$

where b_k and t_k are the k-th outputs of the branch and trunk nets, respectively. We use $p = 128$ output dimensions ($p = 256$ on Controlled Cylinder, Table C.2), 6-layer GELU MLPs for both networks, and dropout 0.1, yielding 3.5M parameters, the smallest model in our benchmark.

Transolver [18]. Standard Transformer attention has a cost of $O(n^2)$ in the number of grid points n. Transolver reduces this cost using *physics-attention*. It compresses the spatial domain into S learned *slice tokens*, each aggregating information from a local neighbourhood of physical points through soft assignment weights. Multi-head attention among these tokens costs $O(S^2)$ and allows long-range information exchange. A final scatter step maps the slice representations back to the original grid.

In our 3D combustion setting we use $S = 16$ slices, 8 attention heads, 1 Transformer layer, and hidden dimension 256, yielding 4.3M parameters. The shallow depth (single layer) is imposed by the memory and compute constraints of the high-dimensional combustion grid ($64 \times 64 \times 20$); its effect on representational capacity is discussed in Section 4.3.

Architectural comparison. Table C.1 provides a side-by-side summary of the four architectures across the dimensions most relevant to sim-to-real transfer.

C.2 Evaluation Metrics

We evaluate model performance from two perspectives: (1) **prediction accuracy**, and (2) **physical consistency**.

Table C.1: Main neural operator architectures.

Architecture	Params	Core Operation	Inductive Bias	Complexity[a]	D.I.[b]	Config (this work)	Fisher[c]
FNO [16]	50.4 M	Spectral (Fourier) convolution	Convolution in frequency domain; high-mode truncation enforces a low-frequency bias	$O(n \log n)$	✓	4 layers, width 64, modes $(4, 12, 16)$[d]	**Low-rank**[1]
CNO [20]	8.0 M	Anti-aliased multi-scale convolution	Alias-free L^2-stable filters; U-Net hierarchy ensures convergence under resolution refinement	$O(n)$	✓	3 enc/dec levels, mult. 32, BatchNorm	**Moderate**[2]
DeepONet [17]	3.5 M	Branch–trunk dot-product sum	Factored basis-function approximation; trunk net queries arbitrary output locations	$O(n_s+n_q)$	✓	p=128, 6-layer MLP, dropout 0.1	**Broad**[3]
Transolver [18]	4.3 M	Physics-attention over learned slice tokens	S learned spatial slices enable global interaction at $O(S^2)$ cost, linear in DOF	$O(S^2)$	✓	1 layer, dim 256, 8 heads, S=16	**Dense**[4]

[a] Dominant cost per forward pass; n: spatial degrees of freedom (DOF), S: number of Transolver slices.
[b] D.I. = Discretisation-invariant: the model can be evaluated at a different spatial resolution than it was trained on.
[c] Fisher structure describes how simulation-critical Fisher information is concentrated across parameters:
[1] **Low-rank**: dominated by a few spectral weight directions.
[2] **Moderate**: spread across multiple resolution scales.
[3] **Broad**: spread over both branch and trunk MLP weights rather than concentrated in one module, yet still captured by a few directions per layer (1–12% of N at $\tau = 0.9$, Figure 5(b)).
[4] **Dense**: concentrated but shallow; few parameters per head due to single-layer design.
[d] Modes $(4, 16, 16)$ for the combustion scenario.

C.2.1 Data-Oriented Metrics

Root mean square error (RMSE). RMSE is the square root of the mean squared difference between predictions and ground truth:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum(\hat{u} - u)^2}, \tag{C.3}$$

where n is the total number of elements.

Relative L_2 error.

$$\text{Rel } L_2 = \frac{1}{B}\sum_{i=1}^{B}\frac{\|\hat{u}_i - u_i\|_2}{\|u_i\|_2}, \tag{C.4}$$

where B is the number of test samples. This normalises the error by the magnitude of the true field.

R^2 score.

$$R^2 = 1 - \frac{\sum(\hat{u} - u)^2}{\sum(u - \bar{u})^2}, \tag{C.5}$$

where $\bar{u}$ is the mean value. A higher R^2 indicates better predictions.

C.2.2 Physics-Oriented Metrics

Spectral error (fRMSE). To evaluate whether the model captures structures at different scales, we transform the data into the frequency domain and compare errors there.

$$\mathrm{fRMSE} = \sqrt{\frac{1}{K}\sum_{\kappa=0}^{K-1} E_\kappa}, \tag{C.6}$$

where E_κ measures the error at frequency level κ.

We also report errors separately for low-, mid-, and high-frequency bands. These bands capture large-scale structures and dominant physical modes, intermediate-scale features, and small-scale fluctuations, respectively. The bandwise errors indicate which scales are affected by fine-tuning.

Relative low-frequency error. Following the RealPDEBench evaluation implementation [6], we compute

$$E_{\mathrm{rel,low}} = \frac{1}{C|\mathcal{K}_{\mathrm{low}}|}\sum_{c=1}^{C}\sum_{\kappa\in\mathcal{K}_{\mathrm{low}}}\frac{\|\mathbf{U}^{\mathrm{pred}}_{\kappa,c} - \mathbf{U}^{\mathrm{ref}}_{\kappa,c}\|_2}{\|\mathbf{U}^{\mathrm{ref}}_{\kappa,c}\|_2}. \tag{C.7}$$

Here $\mathbf{U}^{\mathrm{ref}}_{\kappa,c}$ and $\mathbf{U}^{\mathrm{pred}}_{\kappa,c}$ collect the reference and predicted Fourier coefficients in radial frequency bin κ and channel c across an evaluation batch. The transform spans time and the spatial dimensions; $\mathcal{K}_{\mathrm{low}}$ contains the lowest third of retained bins, and C is the number of channels. We average the metric across evaluation batches. Normalization by the corresponding reference amplitudes gives a dimensionless measure of low-frequency fidelity; lower values are better.

C.3 Experimental Setup

Datasets. We use three scenarios from RealPDEBench, covering fluid flow and combustion. Each dataset is split into training, validation, and test sets using official splits to avoid data leakage.

Pretraining. We retrain the four main architectures on the numerical training split using AdamW and a cosine schedule. The official RealPDEBench settings serve as the reference, with batch sizes and learning rates adjusted for our hardware. Table C.2 gives the actual settings. DPOT-S instead starts from its official checkpoint (Appendix K.2).

Table C.2: Pretraining configurations across architectures and scenarios.

Architecture	Params	LR	Iters	Batch/GPU	GPUs	Eff. batch	Architecture hyperparameters
Scenario 1 · Cylinder flow	grid 128×64, fields u, v, p, horizon $T=20$, ≈5,000 real train samples						
FNO	50.4 M	1×10^{-4}	4,000	16	2	32	`n_layers`=4, `width`=64, modes $(f_x, f_y, f_t) = (4, 12, 16)$
CNO	8.0 M	3×10^{-4}	5,000	4	4	16	`N_layers`=3, `channel_multiplier`=32, `N_res`=1, `N_res_neck`=8, BatchNorm, `latent_lift_proj_dim`=64, LeakyReLU
DeepONet	3.5 M	1×10^{-4}	5,000	4	4	16	p = 128, `dropout`=0.1
Transolver	4.3 M	1.80×10^{-4}	5,000	4	1	4	`n_layers`=1, `n_hidden`=256, `n_head`=8, `slice_num`=16, grid $H\times W\times D = 128\times64\times20$
Scenario 2 · Controlled cylinder	grid 128×64, fields u, v, p, horizon $T=20$, ≈9,500 real train samples						
FNO	50.4 M	1×10^{-4}	4,000	16	4	64	`n_layers`=4, `width`=64, modes $(f_x, f_y, f_t) = (4, 12, 16)$
CNO	8.0 M	3×10^{-4}	5,000	8	4	32	*Same architecture as Scenario 1*
DeepONet	3.5 M	5×10^{-5}	5,000	16	4	64	p = 256, `dropout`=0.1 *(wider trunk for richer actuation input)*
Transolver	4.3 M	1.25×10^{-4}	5,000	4	1	4	`n_layers`=1, `n_hidden`=256, `n_head`=8, `slice_num`=16, grid $H\times W\times D = 64\times128\times10$
Scenario 3 · Turbulent combustion	grid 128×128, fields $T, Y_{\mathrm{OH}}, \ldots$, horizon $T=40$, ≈59,000 real train samples						
FNO	50.4 M	1×10^{-2}	4,000	16	3	48	`n_layers`=4, `width`=64, modes $(f_x, f_y, f_t) = (4, 16, 16)$ *(higher f_y for square grid)*
CNO	8.0 M	3×10^{-4}	5,000	4	3	12	*Same architecture as Scenario 1*
DeepONet	3.5 M	5×10^{-4}	3,000	4	3	12	*Same architecture as Scenario 1*
Transolver	4.3 M	1.75×10^{-4}	5,000	4	1	4	`n_layers`=1, `n_hidden`=256, `n_head`=8, `slice_num`=16, grid $H\times W\times D = 64\times64\times20$

Fine-tuning. All methods use Adam with learning rate 10^{-4}, batch size 4 and 4,000 updates. We validate every 80 updates and report test metrics at the checkpoint with the lowest validation RMSE. Table 1 uses seed 0. Appendix H.1 reports DFT and PhysGuard over seeds 0, 42 and 101. Each fine-tuning run uses one RTX 4090 with 24 GB of memory.

Our shared budget is smaller than the individually tuned RealPDEBench budgets (Table C.3), with 5 to 20 times fewer sample exposures. For DeepONet on Cylinder Flow, the counts are 16,000

and 160,000. The checkpoint and training-budget differences must therefore be considered when comparing absolute results. Appendix H reports checkpoint comparisons and controls using a larger effective batch and the official DeepONet checkpoint.

Table C.3: Fine-tuning settings (batch size, learning rate, updates) in RealPDEBench [6] and this work; ratios compare RealPDEBench to our sample exposures (batch size × updates).

Scenario	Architecture	RealPDEBench	Ours	Ratio
Cylinder Flow	FNO	32, 1×10^{-4}, 4,000	4, 1×10^{-4}, 4,000	8×
	DeepONet	32, 1×10^{-4}, 5,000	4, 1×10^{-4}, 4,000	10×
	CNO	16, 3×10^{-4}, 5,000	4, 1×10^{-4}, 4,000	5×
	Transolver	16, 7×10^{-4}, 5,000	4, 1×10^{-4}, 4,000	5×
Controlled Cylinder	FNO	32, 1×10^{-4}, 4,000	4, 1×10^{-4}, 4,000	8×
	DeepONet	64, 5×10^{-5}, 5,000	4, 1×10^{-4}, 4,000	20×
	CNO	32, 3×10^{-4}, 5,000	4, 1×10^{-4}, 4,000	10×
	Transolver	32, 1×10^{-3}, 5,000	4, 1×10^{-4}, 4,000	10×
Turbulent Combustion	FNO	64, 1×10^{-2}, 2,000	4, 1×10^{-4}, 4,000	8×
	DeepONet	64, 5×10^{-4}, 3,000	4, 1×10^{-4}, 4,000	12×
	CNO	16, 3×10^{-4}, 5,000	4, 1×10^{-4}, 4,000	5×
	Transolver	16, 7×10^{-4}, 5,000	4, 1×10^{-4}, 4,000	5×

Batch size and memory. We use batch size 4 across the 15 configurations, including DPOT-S. Tables C.4 and C.5 give peak memory and feasible batch sizes on a single RTX 4090.

Table C.4: Peak GPU memory on Cylinder Flow (RTX 4090, fp32). OOM: out of memory.

Architecture	Params	Batch 4	Batch 8	Max feasible batch (fp32)
FNO	50.4 M	5.5 GB	9.7 GB	16
CNO	8.0 M	18.4 GB	OOM	4
DeepONet	3.5 M	5.9 GB	11.6 GB	8
Transolver	4.3 M	16.0 GB	OOM	4
DPOT-S	41.3 M	1.2 GB	1.8 GB	128

Table C.5: Maximum feasible batch sizes on one RTX 4090 (24 GB).

Scenario	CNO	Transolver	DeepONet	FNO	DPOT-S
Cylinder Flow (fp32)	4	4	8	16	128
Cylinder Flow (bf16)	8	8	16	16	N/A
Controlled Cylinder (fp32)	8	8	16	16	64
Turbulent Combustion (fp32)	8	8	16	32	64

CNO and Transolver on Cylinder Flow exceed 24 GB at batch 8 in fp32. Batch 4 is the largest shared value that fits all configurations. Table H.6 tests effective batch 16 through gradient accumulation for DeepONet on Cylinder Flow.

PhysGuard settings. The Fisher subspace is estimated using at most 10% of simulation samples. Table D.1 gives the actual sample counts and costs. We retain the top components that explain 90% of variance, up to a maximum of 500. We use projection strength $\alpha = 1$; Appendix G reports sensitivity to α and τ.

PhysGuard memory management. Fisher estimation runs once before fine-tuning. Per-sample gradients are computed at batch size 1 and immediately transferred to CPU memory, where the Gram matrix, eigendecomposition and protected basis are computed. During fine-tuning the basis remains on the CPU, with each layer's vectors loaded onto the GPU when needed. This implementation retains the batch-size limits reported above.

Hardware. The workstation has the following configuration.

- **CPU.** AMD Ryzen Threadripper PRO 5995WX (64 cores / 128 threads)
- **RAM.** 512 GB system memory
- **GPUs.** 4 × NVIDIA GeForce RTX 4090 (24 GB GDDR6X each, 96 GB total GPU memory)
- **Software.** Python 3.10, PyTorch 2.10.0, CUDA 12.8, NVIDIA driver 550.144.03

Pretraining uses PyTorch Distributed Data Parallel with BF16 mixed precision and the GPU counts in Table C.2. Fine-tuning runs use one GPU each, allowing independent experiments to run concurrently.

C.4 Adaptation Methods

Direct fine-tuning (DFT). All parameters are updated freely using real data. This often improves accuracy but may overwrite simulation knowledge.

L_2-SP. Adds a penalty that keeps parameters close to their pretrained values:

$$\mathcal{L} = \mathcal{L}_{\text{real}} + \frac{\lambda}{2}\|\theta - \theta^*\|^2. \tag{C.8}$$

EWC. EWC applies different penalties to different parameters based on their importance:

$$\mathcal{L} = \mathcal{L}_{\text{real}} + \frac{\lambda}{2}\sum_j F_{jj}(\theta_j - \theta_j^*)^2. \tag{C.9}$$

PhysGuard. Instead of adding penalties, PhysGuard directly modifies the gradient: updates are restricted so that they do not change the most important directions learned from simulation.

Additional baselines. Appendix G compares PhysGuard with three further adaptation strategies (LoRA, layer freezing and a matched-rank random subspace, defined there) on Cylinder Flow with DeepONet, all trained under the fine-tuning protocol of Appendix C.3 (4,000 updates, the same optimiser and learning rate, seed 0).

Table C.6: Standard LoRA applicability across five neural operator architectures.

Architecture	Standard LoRA applicability	Main reason
DeepONet	Directly applicable	Primarily uses linear and convolutional layers
CNO	Directly applicable	Primarily uses convolutional layers
Transolver	Directly applicable	Uses attention, MLP, and convolutional layers
DPOT-S	Partially applicable	Spectral and temporal cores contain standalone einsum parameters
FNO	Requires modification	Core parameters are complex-valued spectral tensors

Table C.6 summarises the consequence of the varied parameterisations for LoRA. Standard LoRA applies directly to linear, convolutional and attention layers, which covers DeepONet, CNO and Transolver, whereas the spectral and temporal cores of DPOT-S contain standalone einsum parameters and the core parameters of FNO are complex-valued spectral tensors, both requiring architecture-specific modifications. LoRA has indeed been reported to perform poorly on Fourier neural operators because of a provable approximation bottleneck in spectral layers [47]. GPM [14] faces a similar difficulty, since its activation-based projection is not directly defined for complex spectral and einsum-based parameter blocks. PhysGuard operates on parameter gradients and therefore applies uniformly to every parameter type, including the complex-valued FNO weights treated in Appendix E.

D Fisher Subspace Estimation

This appendix derives the Fisher subspace estimator in Section 3.2.

D.1 Fisher Information and Loss Sensitivity

The empirical FIM measures local sensitivity of the per-sample losses to parameter changes. Formally, for layer m with d_m parameters collected in $\theta^{(m)}$, the empirical FIM at the pretrained weights θ^* is

$$\boldsymbol{F}^{(m)} \;=\; \frac{1}{N}\sum_{i=1}^{N} g_i g_i^\top \;=\; \frac{1}{N}\,\boldsymbol{G}^{(m)\top}\boldsymbol{G}^{(m)} \;\in\; \mathbb{R}^{d_m\times d_m} \tag{D.1}$$

where $g_i = \nabla_{\theta^{(m)}}\ell(\theta^*; x_i, y_i) \in \mathbb{R}^{d_m}$ is the per-sample gradient on the i-th simulation sample, and $\boldsymbol{G}^{(m)} \in \mathbb{R}^{N\times d_m}$ is the matrix stacking all N such gradients row-by-row.

To see why this captures sensitivity, consider a small perturbation $\delta \in \mathbb{R}^{d_m}$ to the weights. The average squared first-order change in per-sample loss is given by

$$\delta^\top \boldsymbol{F}^{(m)}\delta = \frac{1}{N}\sum_{i=1}^{N}\big(\delta^\top g_i\big)^2 \tag{D.2}$$

The leading eigenvectors therefore identify the most loss-sensitive directions on the sampled data. Their association with low-frequency output structure is evaluated in Section 4.2.

D.2 Rank Bound

We collect N simulation samples ($N = 200$ in most configurations; Table D.1), so the gradient matrix $\boldsymbol{G}^{(m)}$ has shape $N \times d_m$. The FIM $\boldsymbol{F}^{(m)} = \frac{1}{N}\boldsymbol{G}^{(m)\top}\boldsymbol{G}^{(m)}$ is a sum of N rank-1 matrices, so its rank is at most N.

Thus, at most N eigenvalues of $\boldsymbol{F}^{(m)}$ are non-zero, regardless of d_m. For FNO with approximately 50M parameters in total (millions in each spectral layer), $N = 200 \ll d_m$, so the physics-critical subspace spans at most 200 directions out of millions.

Write the SVD as $\boldsymbol{G}^{(m)} = \boldsymbol{P}\boldsymbol{\Sigma}\boldsymbol{Q}^\top$ where $\boldsymbol{P} \in \mathbb{R}^{N\times N}$, $\boldsymbol{\Sigma} = \mathrm{diag}(\sigma_1, \ldots, \sigma_N)$, $\boldsymbol{Q} \in \mathbb{R}^{d_m\times N}$. Substituting into Equation (D.1) gives

$$\boldsymbol{F}^{(m)} = \frac{1}{N}\,\boldsymbol{Q}\,\boldsymbol{\Sigma}^2\,\boldsymbol{Q}^\top \tag{D.3}$$

so the columns of $\boldsymbol{Q}$ are eigenvectors of $\boldsymbol{F}^{(m)}$ with eigenvalues σ_j^2/N. The remaining $d_m - N$ directions lie in its null space. This sample-induced rank bound is distinct from concentration of the nonzero spectrum, which determines the selected k_m.

D.3 Gram Matrix Formulation

Forming and eigendecomposing the $d_m \times d_m$ matrix $\boldsymbol{F}^{(m)}$ directly costs $O(d_m^3)$ and is infeasible for d_m in the millions. We instead use the $N \times N$ Gram matrix $\boldsymbol{K} = \boldsymbol{G}\boldsymbol{G}^\top$ in place of the $d_m \times d_m$ matrix $\boldsymbol{G}^\top\boldsymbol{G}$. These two matrices share all their non-zero eigenvalues.

Proposition 1 (Eigenvalue correspondence)**.** *Let $\boldsymbol{G} \in \mathbb{R}^{N\times d_m}$ with $N \ll d_m$. If $v \in \mathbb{R}^N$ is an eigenvector of $\boldsymbol{K} = \boldsymbol{G}\boldsymbol{G}^\top$ with eigenvalue $\lambda \neq 0$, then $\tilde{u} = \boldsymbol{G}^\top v \in \mathbb{R}^{d_m}$ is a non-zero eigenvector of $\boldsymbol{G}^\top\boldsymbol{G}$ with eigenvalue λ, and of $\boldsymbol{F}^{(m)}$ with eigenvalue λ/N.*

Proof. Left-multiplying $\boldsymbol{G}\boldsymbol{G}^\top v = \lambda v$ by $\boldsymbol{G}^\top$ gives

$$\boldsymbol{G}^\top\big(\boldsymbol{G}\boldsymbol{G}^\top v\big) = \boldsymbol{G}^\top(\lambda v) \quad\Longrightarrow\quad \big(\boldsymbol{G}^\top\boldsymbol{G}\big)\big(\boldsymbol{G}^\top v\big) = \lambda\big(\boldsymbol{G}^\top v\big). \tag{D.4}$$

Thus, $\boldsymbol{G}^\top v$ is an eigenvector of $\boldsymbol{G}^\top\boldsymbol{G}$ with eigenvalue λ. It is non-zero because if $\boldsymbol{G}^\top v = 0$, then $\boldsymbol{K}v = \boldsymbol{G}\boldsymbol{G}^\top v = \boldsymbol{G}\cdot 0 = 0$, contradicting $\lambda \neq 0$. □

Instead of decomposing a $d_m \times d_m$ matrix, we only need to decompose the $N \times N$ Gram matrix $\boldsymbol{K} = \boldsymbol{G}\boldsymbol{G}^\top$, which has just $N^2 = 40{,}000$ entries for $N = 200$. The eigendecomposition costs $O(N^3)$ rather than $O(d_m^3)$; the full estimation cost also includes collecting and mapping gradients.

D.4 Subspace Recovery

Given the eigenvectors $\{v_j\}_{j=1}^N$ and eigenvalues $\{\lambda_j\}_{j=1}^N$ of $\boldsymbol{K}$, sorted in descending order, we recover the basis in three steps.

- **Select how many directions to protect.** We retain the smallest k whose cumulative Fisher information reaches the threshold τ,
$$k_m = \min\left\{ k \;\middle|\; \frac{\sum_{j=1}^{k} \lambda_j}{\sum_{j=1}^{N} \lambda_j} \geq \tau \right\}, \quad \tau = 0.9. \tag{D.5}$$
This retains eigenvectors that collectively explain 90% of total Fisher variance.
- **Map back to parameter space.** For each selected $j = 1, \ldots, k_m$, compute the corresponding normalized FIM eigenvector as
$$u_j = \frac{\boldsymbol{G}^{(m)\top} v_j}{\|\boldsymbol{G}^{(m)\top} v_j\|_2} \in \mathbb{R}^{d_m}. \tag{D.6}$$
The normalisation ensures $\|u_j\|_2 = 1$ so the projection operator is well-defined.
- **Assemble the physics-critical basis.** Stack the k_m directions as columns to obtain
$$\boldsymbol{U}^{(m)} = \left[u_1, u_2, \ldots, u_{k_m}\right] \in \mathbb{R}^{d_m \times k_m}. \tag{D.7}$$
The columns of $\boldsymbol{U}^{(m)}$ form an approximately orthonormal set and span the *physics-critical subspace* for layer m. Fine-tuning gradients are projected onto its orthogonal complement.

D.5 Computational Cost

For a layer with d_m parameters, gradient collection requires N forward and backward passes. Constructing the Gram matrix costs $O(N^2 d_m)$, its eigendecomposition costs $O(N^3)$, and recovering k_m basis vectors costs $O(N k_m d_m)$. Storing the gradients and basis requires $O(N d_m)$ and $O(k_m d_m)$ memory, respectively. Thus only the decomposition dimension is independent of d_m.

Estimation is performed once before fine-tuning and takes 2.3 to 32.9 minutes across the tested configurations. Each fine-tuning step adds one projection per layer.

Table D.1 reports the actual sample counts and offline estimation times. The 10% sampling fraction is an upper limit. Eight configurations use $N = 200$, and the others use $N = 1{,}813$, 1,892, 2,000 or 5,886.

Table D.1: Fisher sample counts and offline estimation times.

Scenario	Architecture	N (% of sim.)	Time (min)
Cylinder Flow	FNO	200 (1.1%)	2.9
	CNO	200 (1.1%)	6.1
	DeepONet	200 (1.1%)	5.4
	Transolver	1,813 (10%)	7.8
	DPOT-S	1,813 (10%)	25.4
Controlled Cylinder	FNO	1,892 (10%)	32.9
	CNO	200 (1.1%)	2.6
	DeepONet	200 (1.1%)	2.3
	Transolver	1,892 (10%)	9.9
	DPOT-S	1,892 (10%)	17.6
Turbulent Combustion	FNO	200 (0.34%)	7.4
	CNO	2,000 (3.4%)	32.5
	DeepONet	200 (0.34%)	4.4
	Transolver	5,886 (10%)	28.4
	DPOT-S	200 (0.34%)	3.5

Fisher estimation accounts for 1.2 to 18.4% of total runtime. At $N = 200$ it takes 2.3 to 7.4 minutes and contributes less than 5%. These measurements include the parameter-dependent operations as well as eigendecomposition.

D.6 Sensitivity to Sample Size

We compare protected subspaces and downstream performance at different Fisher sample counts.

Subspace overlap. For DeepONet on Cylinder Flow, we compare each estimate with the $N = 1{,}813$ reference using the mean squared cosine of principal angles. Overlap reaches 0.993 at $N = 200$ and changes little at larger N (Table D.2). The two additional configurations have overlap 0.95.

Table D.2: Subspace overlap measured by mean squared cosines of principal angles.

N (DeepONet, Cylinder Flow)	100	200	500	1,000	1,813 (ref.)
Overlap with $N = 1{,}813$	0.939	0.993	0.994	0.996	1.000

Configuration	N compared	Overlap
CNO, Turbulent Combustion	200 vs. 2,000	0.95
FNO, Cylinder Flow	200 vs. 1,000	0.95

Adaptation performance. Table D.3 reports complete fine-tuning runs with small and large N, keeping other settings fixed within each configuration. RMSE changes little for DPOT-S and CNO and decreases from 0.06036 to 0.05945 for FNO. These controls support smaller sample counts in the tested configurations.

Table D.3: Test RMSE at validation-selected checkpoints for small and large Fisher sample counts N.

Configuration	N compared	Test RMSE
DPOT-S, Controlled Cylinder	200 vs. 1,892	0.01352 vs. 0.01350
CNO, Turbulent Combustion	200 vs. 2,000	0.02685 vs. 0.02685
FNO, Cylinder Flow	200 vs. 1,000	0.06036 vs. 0.05945

E Complex-Valued Weights in FNO

Among the evaluated architectures, only FNO has complex-valued learnable parameters. We describe how these parameters are handled during gradient projection.

E.1 Spectral Weights

FNO's core operation is spectral convolution. At each layer, the input feature map is first transformed into the frequency domain using a Fast Fourier Transform (FFT). FNO then directly multiplies these complex Fourier coefficients by a set of learnable weight matrices to mix information across channels. Because these weight matrices operate on complex numbers, they are themselves stored as complex numbers, specifically in PyTorch's `torch.complex64` format. CNO uses FFT internally for anti-aliasing, but its learnable convolution kernels are still real-valued.

E.2 Complex Gradients

Our entire pipeline, collecting gradients, building the Gram matrix, running the eigendecomposition, and projecting gradients, is built on standard linear algebra over the real numbers. When a weight w is complex, say $w = a + ib$, its gradient is also complex:

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial a} + i \, \frac{\partial \mathcal{L}}{\partial b}. \tag{E.1}$$

We cannot directly stack complex-valued gradients into a real matrix $\boldsymbol{G}$, dot-product them, or apply the Fisher projection without some care.

E.3 Real-Imaginary Concatenation

For a layer with d complex-valued parameters, we treat each complex weight $w_j = a_j + ib_j$ as a pair of real numbers (a_j, b_j). This turns the layer into an equivalent layer with $2d$ real parameters. Concretely, whenever we collect a gradient vector for an FNO layer, we immediately convert it by splitting into real and imaginary parts and concatenating them:

$$g_{\text{real}} = \left[\operatorname{Re}(g_w) \,,\ \operatorname{Im}(g_w) \right] \ \in \ \mathbb{R}^{2d}. \tag{E.2}$$

We build the $N \times N$ Gram matrix from these $\mathbb{R}^{2d}$ vectors, eigendecompose it, and obtain the physics-critical subspace as a set of real directions in $\mathbb{R}^{2d}$. The gradient projection is then applied in the same $\mathbb{R}^{2d}$ space.

After projection, we reconstruct the complex gradient as

$$\hat{\nabla}_W \mathcal{L} = \hat{g}_{1:d} + i \, \hat{g}_{d+1:2d}. \tag{E.3}$$

This is handed back to the optimiser as the (projected) complex gradient, and training continues normally.

E.4 PyTorch and Wirtinger Calculus

Treating a complex weight as two real numbers is not an approximation. It is exactly how PyTorch itself represents complex tensors in memory, and it is consistent with Wirtinger calculus, the standard framework for differentiating functions of complex variables where both the real and imaginary components are free parameters.

F Spectral Probes of Fisher Directions

This appendix reports spectral probes of pretrained operators. The protocol below describes the FNO probe in Figure 3. Table F.1 reports additional results for CNO, DeepONet, and Transolver.

F.1 FNO Probe Procedure

For the FNO probe, let v be a unit-norm direction restricted to the target layer of the pretrained model $f(\theta^*; x)$.

1. **Perturb.** Set $\theta_\epsilon = \theta^* + \epsilon v$ with $\epsilon = 10^{-3}$.
2. **Evaluate.** Apply the original and perturbed models to 40 held-out real validation samples in batches of 4.
3. **Output difference.** Compute $\Delta y_i = f(\theta_\epsilon; x_i) - f(\theta^*; x_i)$ for each sample.
4. **2D FFT.** Average the output difference over time and compute its spatial power spectrum $|\hat{\Delta y}_i(\kappa_x, \kappa_y)|^2$.
5. **Low-frequency energy.** Use modes satisfying $\sqrt{\kappa_x^2 + \kappa_y^2} \le \kappa_{\text{cut}}$, where $\kappa_{\text{cut}} = \text{round}(\kappa_{\max}/3)$ and $\kappa_{\max} = \min(H/2, W/2)$. Their energy fraction is

$$f_{\text{low}} = \frac{\sum_{(\kappa_x,\kappa_y)\in \text{low-}f} |\hat{\Delta y}(\kappa_x, \kappa_y)|^2}{\sum_{(\kappa_x,\kappa_y)} |\hat{\Delta y}(\kappa_x, \kappa_y)|^2}. \tag{F.1}$$

   We average this fraction over the batch.

F.2 FNO Probe Directions

The FNO probe compares the following directions.

- **FIM eigenvectors.** The leading 50 directions, estimated from $N_{\text{FIM}} = 80$ simulation gradients using the Gram-matrix construction (Appendix D).
- **Random directions.** Ten independent Gaussian vectors normalised to unit length. These are not matched to individual eigenvectors.

F.3 FNO Probe Layer

Only FNO's frequency-domain layer `spectral_convs.1.weights1` is perturbed; other layers stay fixed. Complex gradients use concatenated real and imaginary components. This probe uses fixed leading directions; PhysGuard selects each layer's protected subspace by the cumulative-eigenvalue threshold.

F.4 Results across Architectures

Table F.1: Ranges of low-frequency energy fraction f_{low} over top-k FIM and random directions.

Model	Top-k FIM directions	Random directions
FNO	0.9970–0.9999	0.5752–0.8126
CNO	0.9969–0.9980	0.5202–0.7355
DeepONet	0.9966–0.9992	0.6960–0.7991
Transolver	0.9994–0.9999	0.5973–0.7979

In the probes summarized in Table F.1, the leading Fisher directions yield $f_{\text{low}} \ge 0.996$ across the four tested architectures. The association therefore extends beyond FNO in these experiments.

PhysGuard relies on two empirical conditions. The source knowledge to preserve should be concentrated in a low-dimensional Fisher subspace, and its leading directions should affect stable signal structures. In these experiments, the associated output changes are predominantly low-frequency. Whether this protection extends to other signal domains remains untested.

G Additional Baselines and Ablations

All comparisons in this appendix use Cylinder Flow with DeepONet, 4,000 updates, the same optimiser and learning rate, and seed 0 (Appendix C.3). Unless varied in an ablation, PhysGuard uses $\alpha = 1$, $\tau = 0.9$, and $N = 200$ Fisher samples.

G.1 Additional Adaptation Baselines

LoRA [15] uses rank $r = 8$ and scaling $\alpha_{\text{LoRA}} = 16$, without additional tuning. **Random subspace** replaces each layer's Fisher basis with a fixed random orthonormal basis of the same dimension, retaining the projection in Equation (5). **Layer freezing** fixes part of the network and fine-tunes the remaining layers.

Table C.6 summarises the applicability of standard LoRA across the operator architectures.

Table G.1: Additional baselines for DeepONet on Cylinder Flow. Best values are bold.

Method	RMSE ↓	R^2 ↑	fRMSE ↓	Low-f ↓	Arch.-agnostic
DFT	0.09309	0.5371	0.01507	0.02527	Yes
LoRA ($r = 8$)	0.08654	0.5963	0.01424	0.02278	No
Random subspace	0.09114	0.5465	0.01483	0.02441	Yes
Layer freezing	0.08975	0.5692	0.01462	0.02403	Yes
PhysGuard	**0.07435**	**0.7048**	**0.01246**	**0.01957**	Yes

PhysGuard has the best values on all four metrics in Table G.1. The matched-rank random projection remains close to DFT, supporting the role of Fisher-based direction selection. Appendix I extends this control to four architectures and reports simulation-domain forgetting.

G.2 Projection Strength α

In Equation (5), $\alpha = 0$ recovers DFT and $\alpha = 1$ removes the gradient component in the protected subspace. Table G.2 varies α with $\tau = 0.9$ and $N = 200$ fixed.

Table G.2: Projection-strength (α) ablation for DeepONet on Cylinder Flow. Best values are bold.

α	Method	RMSE ↓	R^2 ↑	fRMSE ↓	Low-f ↓
0	DFT	0.09309	0.5371	0.01507	0.02527
0.3	PhysGuard (weak)	0.08242	0.6328	0.01357	0.02365
0.5	PhysGuard (moderate)	0.08236	0.6351	0.01363	0.02367
0.7	PhysGuard (strong)	0.08188	0.6403	0.01345	0.02358
1.0	PhysGuard (full)	**0.07435**	**0.7048**	**0.01246**	**0.01957**

Full projection has the best values on all four metrics in this setting. The intermediate strengths $\alpha \in \{0.3, 0.5, 0.7\}$ differ by at most 0.00054 RMSE.

G.3 Variance Threshold τ

The threshold τ selects the smallest per-layer k whose cumulative eigenvalue ratio reaches τ (Equation (4)). Table G.3 varies τ with $\alpha = 1$ and $N = 200$ fixed. The rank cap of 200 is inactive, with the largest observed $k = 24$.

Table G.3: Fisher-threshold (τ) ablation for DeepONet on Cylinder Flow. Best values are bold.

τ	RMSE↓	R^2↑	fRMSE↓	Low-f↓	Per-layer k (range)
0.80	0.07568	0.6918	0.01274	0.02018	$[1, 3]$
0.85	0.07482	0.7013	0.01254	0.01978	$[1, 5]$
0.90 (default)	**0.07435**	**0.7048**	**0.01246**	**0.01957**	$[1, 8]$
0.95	0.07522	0.6952	0.01266	0.02005	$[2, 12]$
0.99	0.07785	0.6728	0.01291	0.02128	$[3, 24]$

The default $\tau = 0.90$ has the best values on all four metrics. All tested thresholds outperform DFT, although increasing τ to 0.99 reduces the gain and protects more directions.

G.4 Scope of the Random-Subspace Control

With $N \ll d$, the empirical Fisher has rank at most N, leaving a null space of dimension at least $d - N$. The matched-rank random control tests arbitrary direction selection. It is not a separate experiment on the bottom Fisher eigenspace.

H Additional Results and Experimental Controls

We report additional relative low-frequency error results and examine variation across seeds and controls for training budget, batch size and pretrained checkpoint. Unless stated otherwise, we use the protocol in Appendix C.3 and report test-set metrics at the checkpoint with the lowest validation RMSE.

H.1 Seed Variability

Table H.1: DFT and PhysGuard across seeds (mean ± standard deviation where reported).

Architecture	Method	Cylinder Flow		Controlled Cylinder		Turbulent Combustion	
		RMSE ↓	Low-f ↓	RMSE ↓	Low-f ↓	RMSE ↓	Low-f ↓
FNO	DFT	0.0648 ± 0.0013	0.0165 ± 0.0004	**0.0099 ± 0.0002**†	**0.0011**	0.0256 ± 0.0006	0.0042 ± 0.0001
	PhysGuard	**0.0606 ± 0.0013**	**0.0113 ± 0.0002**	0.0100 ± 0.0002	**0.0011**	**0.0254 ± 0.0005**†	**0.0041 ± 0.0001**†
CNO	DFT	0.0498 ± 0.0006	0.0159 ± 0.0002	**0.0089 ± 0.0002**	**0.0009**	0.0269 ± 0.0004	0.0048 ± 0.0001
	PhysGuard	**0.0464 ± 0.0007**	**0.0115 ± 0.0003**	**0.0089 ± 0.0002**	**0.0009**	**0.0266 ± 0.0005**†	**0.0046 ± 0.0001**
DeepONet	DFT	0.0928 ± 0.0014	0.0253 ± 0.0004	0.0302 ± 0.0006	0.0045 ± 0.0001	0.0277 ± 0.0006	0.0045 ± 0.0001
	PhysGuard	**0.0744 ± 0.0011**	**0.0197 ± 0.0004**	**0.0301 ± 0.0006**†	**0.0044 ± 0.0001**†	**0.0275 ± 0.0004**†	**0.0044 ± 0.0001**†
Transolver	DFT	0.1120 ± 0.0022	0.0335 ± 0.0008	0.0190 ± 0.0003	0.0025	0.0401 ± 0.0009	0.0083 ± 0.0001
	PhysGuard	**0.1067 ± 0.0018**	**0.0309 ± 0.0006**	**0.0185 ± 0.0004**	**0.0024**	**0.0384 ± 0.0007**	**0.0076 ± 0.0002**

Tables H.2–H.4 compare DFT and PhysGuard across seeds (0, 42, 101), with RMSE and Low-f summarized in Table H.1. The largest differences occur on Cylinder Flow, where every gain exceeds the seed-to-seed spread. On Controlled Cylinder and Turbulent Combustion, several smaller differences fall within one standard deviation (marked with †). Entries without a reported standard deviation are shown as means only.

Table H.2: Cylinder Flow results across seeds (mean ± SD; best means in bold).

Architecture	Method	RMSE ↓	R^2 ↑	fRMSE ↓	Low-f ↓
FNO	DFT	0.0648 ± 0.0013	0.7757 ± 0.0090	0.0105 ± 0.0001	0.0165 ± 0.0004
	PhysGuard	**0.0606 ± 0.0013**	**0.8041 ± 0.0085**	**0.0092 ± 0.0002**	**0.0113 ± 0.0002**
CNO	DFT	0.0498 ± 0.0006	0.8675 ± 0.0033	0.0086 ± 0.0002	0.0159 ± 0.0002
	PhysGuard	**0.0464 ± 0.0007**	**0.8850 ± 0.0037**	**0.0076 ± 0.0002**	**0.0115 ± 0.0003**
DeepONet	DFT	0.0928 ± 0.0014	0.5400 ± 0.0140	0.0151 ± 0.0004	0.0253 ± 0.0004
	PhysGuard	**0.0744 ± 0.0011**	**0.7045 ± 0.0091**	**0.0125 ± 0.0003**	**0.0197 ± 0.0004**
Transolver	DFT	0.1120 ± 0.0022	0.3298 ± 0.0267	0.0163 ± 0.0003	0.0335 ± 0.0008
	PhysGuard	**0.1067 ± 0.0018**	**0.3924 ± 0.0201**	**0.0157 ± 0.0003**	**0.0309 ± 0.0006**

Table H.3: Controlled Cylinder results across seeds (mean ± SD where available; best means in bold).

Architecture	Method	RMSE ↓	R^2 ↑	fRMSE ↓	Low-f ↓
FNO	DFT	**0.0099 ± 0.0002**	**0.9804 ± 0.0007**	**0.0012**	**0.0011**
	PhysGuard	0.0100 ± 0.0002	0.9802 ± 0.0010	**0.0012**	**0.0011**
CNO	DFT	**0.0089 ± 0.0002**	0.9843 ± 0.0005	0.0011	**0.0009**
	PhysGuard	**0.0089 ± 0.0002**	**0.9844 ± 0.0007**	**0.0010**	**0.0009**
DeepONet	DFT	0.0302 ± 0.0006	0.8189 ± 0.0071	**0.0057 ± 0.0001**	0.0045 ± 0.0001
	PhysGuard	**0.0301 ± 0.0006**	**0.8199 ± 0.0070**	**0.0057 ± 0.0001**	**0.0044 ± 0.0001**
Transolver	DFT	0.0190 ± 0.0003	0.9285 ± 0.0021	0.0027	0.0025
	PhysGuard	**0.0185 ± 0.0004**	**0.9320 ± 0.0027**	**0.0025**	**0.0024**

Table H.4: Turbulent Combustion results across seeds (mean ± SD where available; best means in bold).

Architecture	Method	RMSE ↓	R^2 ↑	fRMSE ↓	Low-f ↓
FNO	DFT	0.0256 ± 0.0006	0.6383 ± 0.0159	**0.0033**	0.0042 ± 0.0001
	PhysGuard	**0.0254 ± 0.0005**	**0.6437 ± 0.0130**	**0.0033**	**0.0041 ± 0.0001**
CNO	DFT	0.0269 ± 0.0004	0.6000 ± 0.0106	0.0036 ± 0.0001	0.0048 ± 0.0001
	PhysGuard	**0.0266 ± 0.0005**	**0.6104 ± 0.0140**	**0.0035 ± 0.0001**	**0.0046 ± 0.0001**
DeepONet	DFT	0.0277 ± 0.0006	0.5762 ± 0.0177	**0.0037 ± 0.0001**	0.0045 ± 0.0001
	PhysGuard	**0.0275 ± 0.0004**	**0.5829 ± 0.0107**	**0.0037 ± 0.0001**	**0.0044 ± 0.0001**
Transolver	DFT	0.0401 ± 0.0009	0.1123 ± 0.0383	0.0053 ± 0.0001	0.0083 ± 0.0001
	PhysGuard	**0.0384 ± 0.0007**	**0.1874 ± 0.0300**	**0.0051 ± 0.0001**	**0.0076 ± 0.0002**

Several gains on Controlled Cylinder and Turbulent Combustion are comparable to the reported standard deviations. These results show a favourable overall trend, with less conclusive differences in those scenarios.

H.2 Validation-Selected and Final Checkpoints

Table H.5 compares the checkpoint with the lowest validation RMSE and the final checkpoint (step 4,000) for DFT, EWC and L_2-SP over all twelve main configurations at seed 0. Validation is performed every 80 updates, yielding 50 candidate checkpoints; the "Selected iter." column gives the selected step and its percentage of the 4,000-update budget. RMSE_{sel} and RMSE_{4000} report the corresponding test RMSE values on the same test set for the same training run.

Table H.5: Baseline test RMSE at validation-selected and final checkpoints.

Architecture	Method	Cylinder Flow			Controlled Cylinder			Turbulent Combustion		
		Selected iter. (%)	RMSE_{sel}	RMSE_{4000}	Selected iter. (%)	RMSE_{sel}	RMSE_{4000}	Selected iter. (%)	RMSE_{sel}	RMSE_{4000}
FNO	DFT	160 (4%)	0.06487	0.06738	3680 (92%)	0.00997	0.00998	3440 (86%)	0.02548	0.02553
	EWC	160 (4%)	0.06317	0.06971	3440 (86%)	0.00997	0.01001	3920 (98%)	0.02544	0.02547
	L_2-SP	160 (4%)	0.06556	0.06718	3680 (92%)	0.01191	0.01192	3600 (90%)	0.02779	0.02784
CNO	DFT	1120 (28%)	0.04988	0.06144	3760 (94%)	0.00885	0.00889	3520 (88%)	0.02688	0.02690
	EWC	1520 (38%)	0.04855	0.05311	3600 (90%)	0.00886	0.00888	3760 (94%)	0.02670	0.02671
	L_2-SP	1120 (28%)	0.04915	0.06046	3920 (98%)	0.00936	0.00938	3600 (90%)	0.02685	0.02689
DeepONet	DFT	560 (14%)	0.09309	0.09426	3360 (84%)	0.03004	0.03015	3680 (92%)	0.02757	0.02758
	EWC	2720 (68%)	0.09055	0.09607	3520 (88%)	0.03006	0.03013	3680 (92%)	0.02745	0.02750
	L_2-SP	2560 (64%)	0.09121	0.09237	3280 (82%)	0.03223	0.03239	3760 (94%)	0.02805	0.02810
Transolver	DFT	1520 (38%)	0.11236	0.11339	3840 (96%)	0.01898	0.01900	3520 (88%)	0.03995	0.04002
	EWC	1760 (44%)	0.11330	0.11401	3360 (84%)	0.01898	0.01899	3920 (98%)	0.04070	0.04074
	L_2-SP	1920 (48%)	0.11361	0.11472	3760 (94%)	0.01920	0.01921	3920 (98%)	0.04087	0.04091

The validation-selected checkpoints have lower test RMSE than the final checkpoints in these runs, with larger absolute RMSE differences on Cylinder Flow than on the other two scenarios. Table 1 reports test metrics at the validation-selected checkpoints. These comparisons assess checkpoint choice within the fixed budget; they do not establish convergence or performance under a longer training schedule.

H.3 Effective Batch-Size Control

For DeepONet on Cylinder Flow, we repeat all four methods from the same pretrained checkpoint with effective batch size 16, using four accumulation steps of batch size 4. Learning rate, optimizer, update count, validation protocol and seed 0 remain unchanged. Sample exposures increase from 16,000 to 64,000 (Table H.6).

Table H.6: DeepONet batch-size control on Cylinder Flow (seed 0; effective b=16 via accumulation). Best minimises validation RMSE, final denotes step 4,000, and b=16 uses best checkpoints; bold marks column minima.

Method	RMSE ↓			Low-f ↓	
	best (b=4)	final (b=4)	b=16	b=4	b=16
DFT	0.09309	0.09426	0.09091	0.02527	0.02473
EWC	0.09055	0.09607	0.08868	0.02599	0.02552
L_2-SP	0.09121	0.09237	0.08950	0.02627	0.02583
PhysGuard	**0.07435**	**0.07512**	**0.07308**	**0.01957**	**0.01941**

For DeepONet on Cylinder Flow, the larger effective batch modestly improves all four methods while preserving their order (PhysGuard < EWC < L_2-SP < DFT). PhysGuard retains its advantage in RMSE and Low-f error under both tested settings.

H.4 Official Pretrained Checkpoint Control

Our retrained checkpoints and smaller fine-tuning budget differ from the original RealPDEBench settings. For DeepONet on Cylinder Flow, the published DFT results are RMSE 0.0661 and R^2 0.718, compared with 0.09309 and 0.5371 in Table 1. We therefore also fine-tune the official pretrained checkpoint under our shared protocol (Table H.7).

Table H.7: Fine-tuning the official RealPDEBench DeepONet checkpoint on Cylinder Flow.

Method	RMSE ↓	R^2 ↑	fRMSE ↓	Low-f ↓
DFT (official checkpoint)	0.06613	0.71811	0.01067	0.02661
PhysGuard (official checkpoint)	**0.05360**	**0.81450**	**0.00880**	**0.02267**
Relative change (DFT → PhysGuard)	−18.9%	+13.4%	−17.5%	−14.8%

With the official checkpoint, DFT closely reproduces the published RMSE and R^2. PhysGuard improves all four metrics, including an 18.9% RMSE reduction. This control supports its relative advantage for this checkpoint and configuration.

H.5 Relative Low-Frequency Error

Table H.8 extends the Cylinder Flow comparison in Figure 5(a) to all three scenarios.

Table H.8: Relative low-frequency error across architectures and scenarios.

Scenario	Architecture	Relative low-frequency error ↓			Relative change		
		Pretrained	DFT	PhysGuard	PhysGuard vs DFT	PhysGuard vs Pretrained	DFT vs Pretrained
Cylinder Flow	FNO	0.690	0.794	0.509	−35.9%	−26.1%	+15.1%
	CNO	0.532	0.657	0.386	−41.3%	−27.4%	+23.6%
	DeepONet	0.862	0.837	0.724	−13.6%	−16.0%	−2.8%
	Transolver	0.824	0.711	0.651	−8.4%	−21.0%	−13.7%
Controlled Cylinder	FNO	0.266	0.057	0.058	+0.7%	−78.3%	−78.5%
	CNO	0.103	0.050	0.048	−4.4%	−53.6%	−51.5%
	DeepONet	0.623	0.221	0.219	−0.9%	−64.8%	−64.5%
	Transolver	0.426	0.117	0.114	−2.1%	−73.2%	−72.7%
Turbulent Combustion	FNO	0.566	0.297	0.294	−1.2%	−48.1%	−47.5%
	CNO	0.725	0.339	0.326	−4.0%	−55.1%	−53.2%
	DeepONet	0.651	0.322	0.318	−1.4%	−51.2%	−50.5%
	Transolver	0.763	0.587	0.540	−7.9%	−29.2%	−23.1%

Errors are shown to three decimal places. Percentage changes are computed from unrounded errors relative to the second named method; negative values indicate lower error.

PhysGuard has lower relative low-frequency error than the pretrained model in all twelve configurations and lower error than DFT in eleven. The exception is FNO on Controlled Cylinder, where its error is 0.7% higher than DFT.

I Subspace Selection and Orthogonality

We test how subspace selection affects preservation, whether parameter constraints reduce output drift, and whether strict parameter orthogonality provides additional benefit. All experiments use Cylinder Flow with FNO, CNO, DeepONet and Transolver, following the same fine-tuning protocol (4,000 steps, the same optimiser and learning rate, seed 0). Orthogonality of projected gradients does not guarantee orthogonality of optimiser steps or output changes.

I.1 Matched-Rank Random-Subspace Control

We replace each layer's Fisher basis $\boldsymbol{U}^{(m)}$ with a fixed random orthonormal basis of the same dimension k_m. The pretrained model, projection, training steps and learning rate remain unchanged. This extends the DeepONet control in Table G.1 to four architectures. We measure simulation-domain forgetting by

$$\text{Forgetting} = \frac{\text{RMSE}_{\text{sim}}^{\text{after FT}} - \text{RMSE}_{\text{sim}}^{\text{pretrained}}}{\text{RMSE}_{\text{sim}}^{\text{pretrained}}}, \tag{I.1}$$

using the same simulation test set before and after fine-tuning. Lower values indicate better retention.

Table I.1: Matched-rank random-subspace control on Cylinder Flow. Best values are bold.

Architecture	Method	Real-domain RMSE ↓	Low-f ↓	Sim-domain forgetting (%) ↓
FNO	DFT	0.06487	0.01651	15.3
	Random subspace	0.06491	0.01648	15.5
	PhysGuard	**0.06036**	**0.01131**	**6.1**
CNO	DFT	0.04988	0.01585	12.8
	Random subspace	0.04983	0.01591	12.6
	PhysGuard	**0.04635**	**0.01148**	**5.2**
DeepONet	DFT	0.09309	0.02527	28.7
	Random subspace	0.09114	0.02441	27.5
	PhysGuard	**0.07435**	**0.01957**	**13.4**
Transolver	DFT	0.11236	0.03349	22.1
	Random subspace	0.11229	0.03354	21.8
	PhysGuard	**0.10695**	**0.03097**	**11.8**

The random-subspace results are close to DFT, whereas PhysGuard lowers simulation-domain forgetting, real-domain RMSE and low-frequency error on all four architectures (Table I.1). Under this matched control, the results support the role of Fisher-based direction selection in preservation.

I.2 Output-Space Probe and Leakage

Layer-wise parameter orthogonality need not translate to output orthogonality. We therefore evaluate output changes on held-out simulation samples.

Probe definition. We slightly perturb the two largest protected tensors along their dominant Fisher directions. Orthonormalising the resulting output changes gives a basis $\boldsymbol{Q}$ with 13–40 directions. For an update with output change Δf on the same inputs, we define

$$\text{Leakage}(\Delta f) = \frac{\|\boldsymbol{Q}^\top \Delta f\|_2^2}{\|\Delta f\|_2^2}, \tag{I.2}$$

which measures overlap with these sampled output modes. Perturbations along protected Fisher directions place 96–100% of their output-change energy inside $\boldsymbol{Q}$. This probe measures preservation relative to $\boldsymbol{Q}$ rather than every physical property of the output.

Output changes after fine-tuning. Table I.2 compares total prediction changes and their protection-related components overlapping $\boldsymbol{Q}$. Both are measured on the same simulation inputs and normalised by the corresponding DFT value.

Table I.2: Output changes on Cylinder Flow, normalized to DFT (= 1).

Model	Total output change	Protection-related change
FNO	0.78	0.71
CNO	0.85	0.75
DeepONet	0.80	0.49
Transolver	0.83	0.50

The protection-related change decreases more than the total change on all four architectures. For DeepONet, the reductions are 51% and 20%, respectively. This indicates preferential suppression of changes along the probed output modes.

Output leakage. We compare random updates orthogonal to $\boldsymbol{U}^{(m)}$ in every layer, the actual PhysGuard update, and DFT. The random updates test whether layer-wise parameter orthogonality persists at the output. Their ranges reflect repeated sampling.

Table I.3: Leakage into sampled output modes on Cylinder Flow; ranges are over repeated random updates.

Model	Layer-wise-orthogonal random	PhysGuard update	Direct fine-tuning
FNO	0.004–0.012	0.005	0.006
CNO	0.15–0.24	0.14	0.18
DeepONet	0.35–0.85	0.28	0.76
Transolver	0.18–0.44	0.20	0.56

Nonzero leakage from the random updates confirms that the layer-wise constraint does not guarantee output orthogonality. PhysGuard has lower leakage than DFT in all four cases, although leakage remains nonzero (Table I.3).

I.3 Parameter Changes and Output Drift

Each run saves a checkpoint every 80 steps, giving 50 checkpoints over 4,000 steps.

Cumulative parameter change. Relative to the pretrained parameters $\theta_0 = \theta^*$, we compute

$$\Delta\theta_t = \theta_t - \theta_0, \qquad r_t = \frac{\|\boldsymbol{U}^\top \Delta\theta_t\|_2^2}{\|\Delta\theta_t\|_2^2}, \tag{I.3}$$

where $\boldsymbol{U}$ collects the per-layer bases block-diagonally. The ratio is the protected share of the cumulative parameter change and is reported in percent.

Table I.4: Protected share r_t of cumulative parameter changes across 50 PhysGuard checkpoints on Cylinder Flow.

Architecture	Mean (%) ↓	Max (%) ↓	Final (%) ↓
FNO	0.06	0.15	0.04
CNO	0.09	0.21	0.07
DeepONet	0.33	0.62	0.27
Transolver	0.87	1.45	0.81

The protected share remains below 1.5% at all sampled checkpoints (Table I.4). These small but nonzero values also distinguish gradient projection from a strict constraint on the accumulated parameter change.

Output drift. On a fixed simulation evaluation set X_{sim}, we compute

$$D_t = \frac{\|f_{\theta_t}(X_{\text{sim}}) - f_{\theta_0}(X_{\text{sim}})\|_2}{\|f_{\theta_0}(X_{\text{sim}})\|_2}, \qquad R_t = \frac{D_t^{\text{PhysGuard}}}{D_t^{\text{DFT}}}, \tag{I.4}$$

where $R_t < 1$ indicates less output drift under PhysGuard than DFT.

Table I.5: Function-space drift relative to DFT on Cylinder Flow (mean over 50 checkpoints and final value).

Architecture	Checkpoints with $R_t < 1$	Mean $R_t \downarrow$	Final $R_t \downarrow$
FNO	30/50	0.91	0.78
CNO	43/50	0.78	0.85
DeepONet	46/50	0.73	0.80
Transolver	38/50	0.83	0.83

PhysGuard has less output drift at most checkpoints, with mean and final ratios below one on all four architectures (Table I.5). This reduction does not hold at every checkpoint. The final ratios match the total output changes in Table I.2.

I.4 Post-hoc Orthogonality Correction

At the best-validation checkpoint, we apply a one-time correction

$$\theta_{\text{corr}} = \theta_0 + (\boldsymbol{I} - \boldsymbol{U}\boldsymbol{U}^\top)(\theta_{\text{FT}} - \theta_0), \tag{I.5}$$

independently to each layer. This removes the protected component of the cumulative parameter change. Table I.6 reports the result for PhysGuard and DFT.

Table I.6: Post-hoc orthogonality correction on Cylinder Flow at the best-validation checkpoint; "Protected" is the protected share of cumulative parameter change.

Architecture	Training	Correction	Protected before $\downarrow$	Protected after $\downarrow$	Real RMSE $\downarrow$	Forgetting (%) $\downarrow$
FNO	DFT	None	7.6%	—	0.06487	15.3
	DFT	Fisher-$\boldsymbol{U}$	7.6%	≈ 0	0.06522	14.9
	PhysGuard	None	0.05%	—	0.06036	6.1
	PhysGuard	Fisher-$\boldsymbol{U}$	0.05%	≈ 0	0.06037	6.1
CNO	DFT	None	5.8%	—	0.04988	12.8
	DFT	Fisher-$\boldsymbol{U}$	5.8%	≈ 0	0.05021	12.4
	PhysGuard	None	0.08%	—	0.04635	5.2
	PhysGuard	Fisher-$\boldsymbol{U}$	0.08%	≈ 0	0.04636	5.2
DeepONet	DFT	None	11.3%	—	0.09309	28.7
	DFT	Fisher-$\boldsymbol{U}$	11.3%	≈ 0	0.09372	28.0
	PhysGuard	None	0.31%	—	0.07435	13.4
	PhysGuard	Fisher-$\boldsymbol{U}$	0.31%	≈ 0	0.07436	13.4
Transolver	DFT	None	9.1%	—	0.11236	22.1
	DFT	Fisher-$\boldsymbol{U}$	9.1%	≈ 0	0.11283	21.5
	PhysGuard	None	0.92%	—	0.10695	11.8
	PhysGuard	Fisher-$\boldsymbol{U}$	0.92%	≈ 0	0.10696	11.8

The correction barely changes PhysGuard's reported RMSE and leaves its forgetting unchanged. For DFT, it removes the protected parameter component but only slightly reduces forgetting and slightly increases real-domain RMSE. Strict parameter orthogonality therefore does not recover function-space preservation in this experiment.

J Qualitative Visualisations

In this section, we provide the full visualization of results for FNO, CNO, DeepONet, and Transolver across all three datasets, complementing the quantitative results reported in Table 1.

J.1 Cylinder Flow

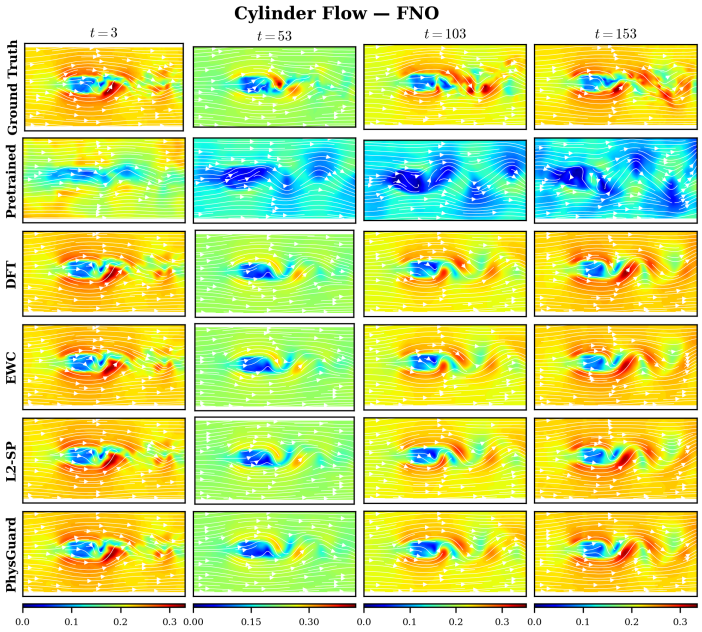


Figure J.1: Cylinder Flow – FNO predictions.

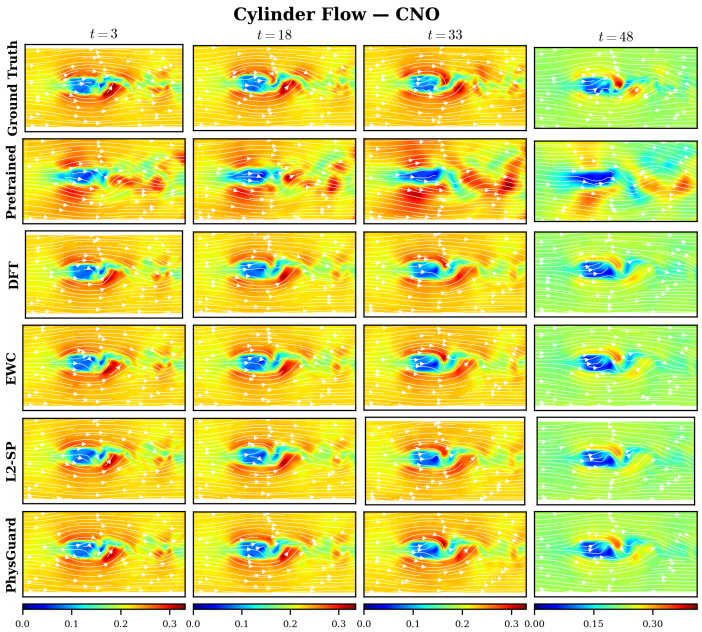


Figure J.2: Cylinder Flow – CNO predictions.

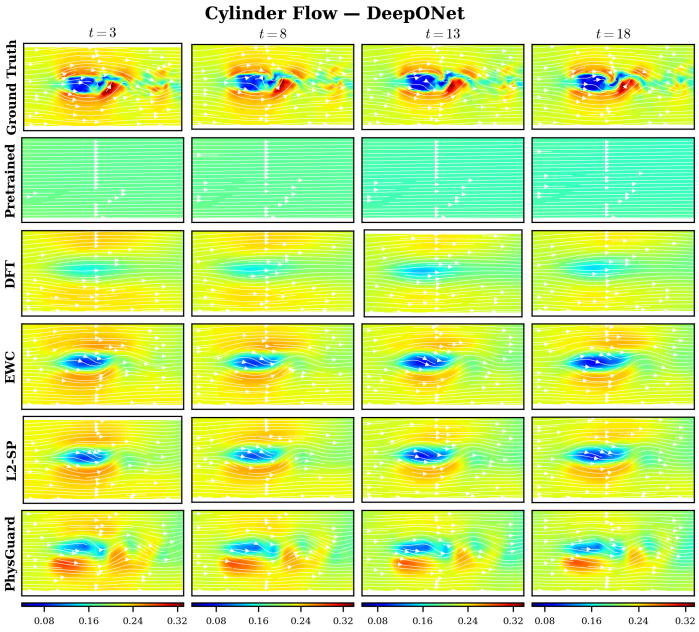


Figure J.3: Cylinder Flow – DeepONet predictions.

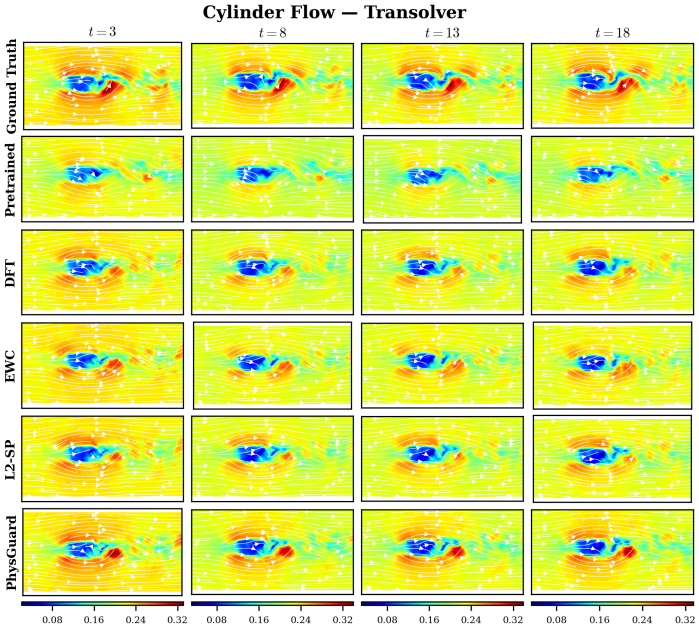


Figure J.4: Cylinder Flow – Transolver predictions.

J.2 Controlled Cylinder

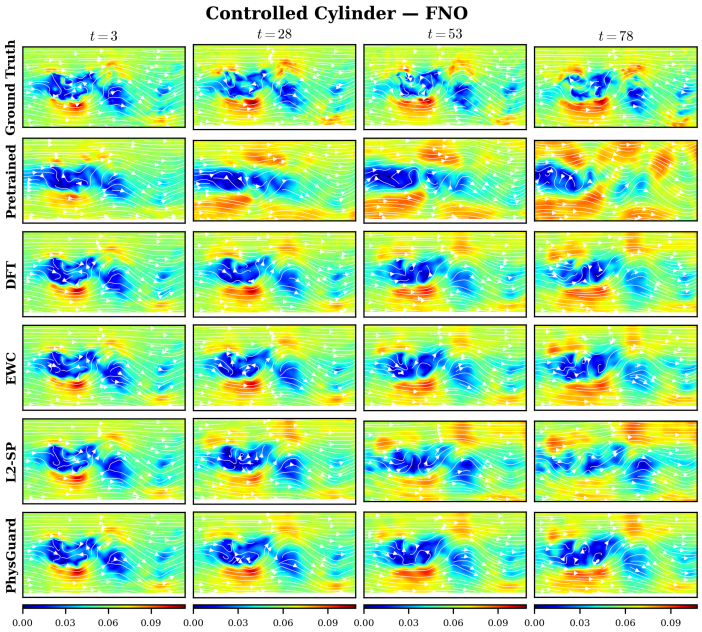


Figure J.5: Controlled Cylinder Flow – FNO predictions.

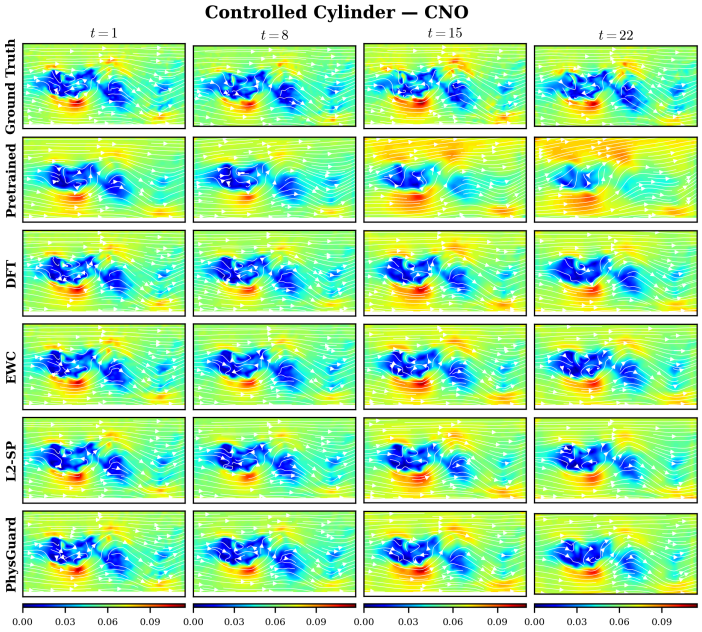


Figure J.6: Controlled Cylinder Flow – CNO predictions.

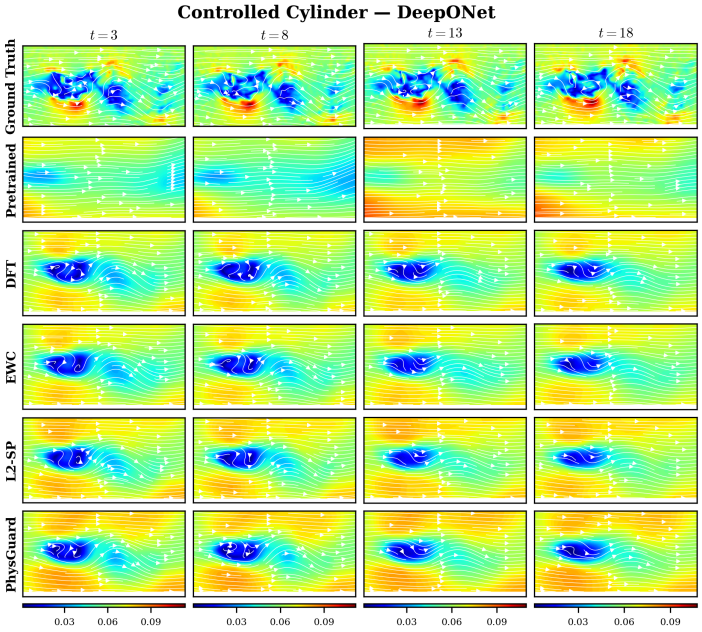


Figure J.7: Controlled Cylinder Flow – DeepONet predictions.

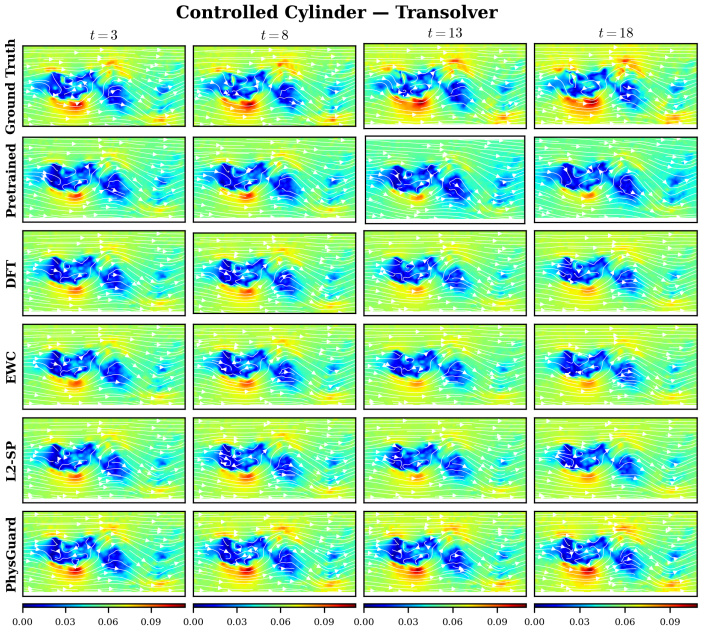


Figure J.8: Controlled Cylinder Flow – Transolver predictions.

J.3 Turbulent Combustion

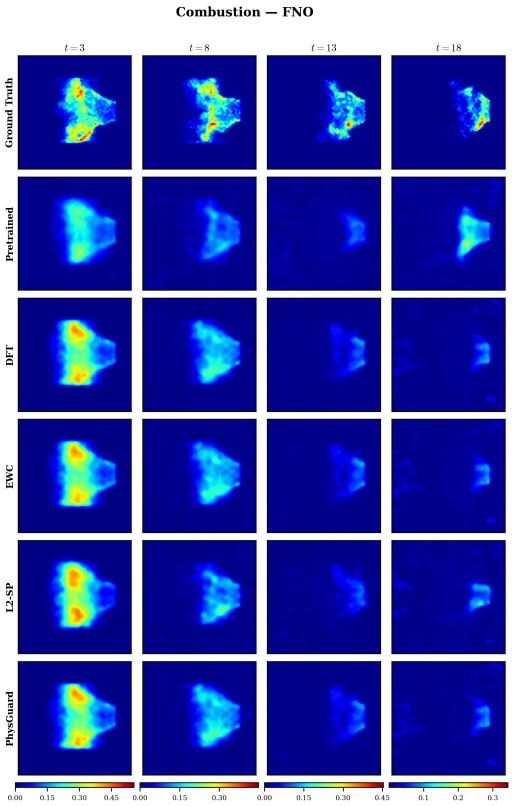


Figure J.9: Turbulent Combustion – FNO predictions.

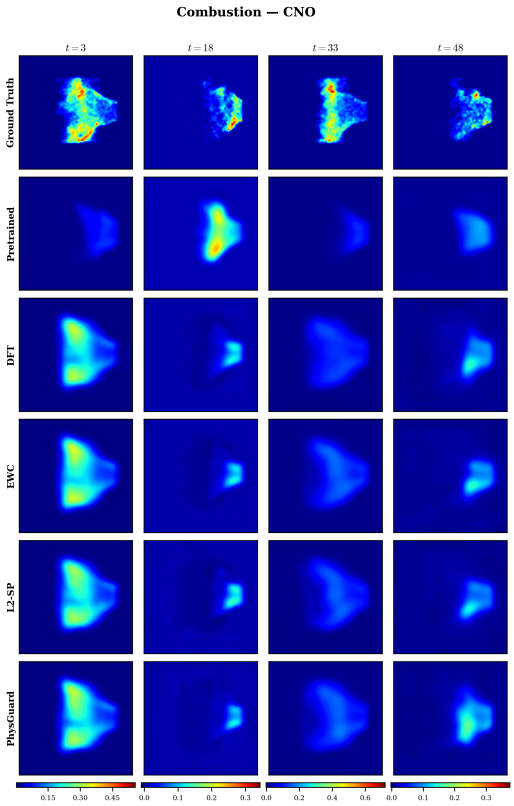


Figure J.10: CNO predictions on Turbulent Combustion.

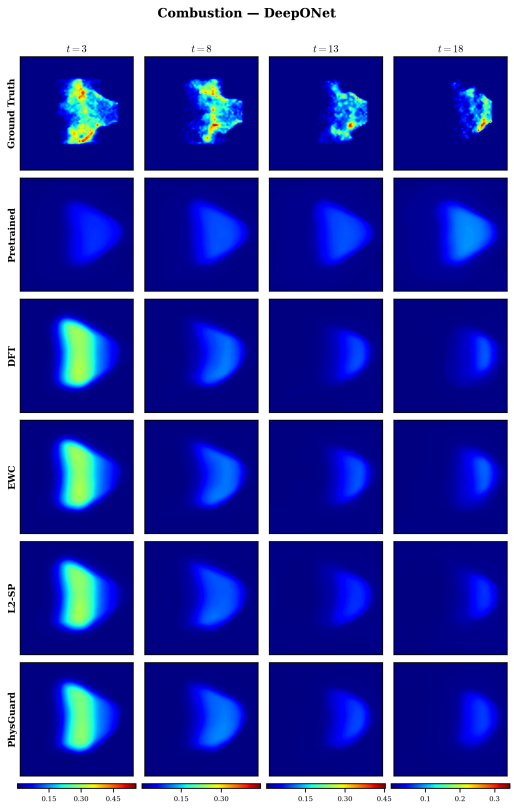


Figure J.11: DeepONet predictions on Turbulent Combustion.

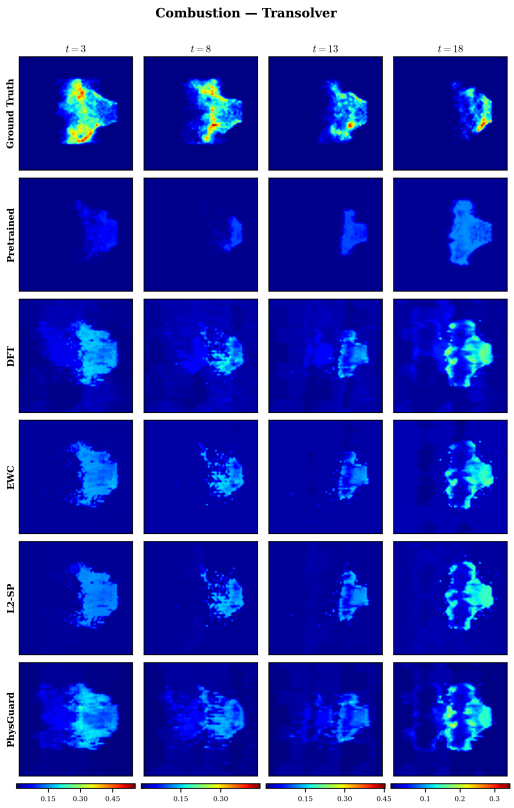


Figure J.12: Transolver predictions on Turbulent Combustion.

K DPOT-S as a Negative Case

PhysGuard does not consistently improve performance on the foundation-scale operator DPOT-S [21], as noted in Section 5. This appendix examines this case in relation to the method's assumption of a *low-rank* Fisher spectrum.

K.1 Architecture

Denoising Pre-training Operator Transformer (DPOT) combines FNO-style spectral feature extraction with a scalable Transformer backbone. Input fields are projected onto a spectral basis, and the resulting tokens are processed by Diffusion-Transformer (DiT) style attention blocks [59]. A shared backbone is pretrained across multiple PDE datasets and then fine-tuned for each task.

We use DPOT-S with 6 Transformer blocks, hidden dimension 1024, 8 attention heads and 41.3M parameters. Its Fisher spectrum at the checkpoint used for adaptation is less concentrated than those of the four main architectures (Appendix K.3).

K.2 Pretraining Configuration

Table K.1 reports the pretraining hyperparameters used for DPOT-S on each of the three RealPDEBench scenarios. The overall protocol (AdamW with cosine annealing, BF16 AMP, DDP across 4 RTX 4090 GPUs) matches the one described in Appendix C.3; only the architecture-specific and scenario-specific values differ.

Table K.1: DPOT-S pretraining settings for the three RealPDEBench scenarios.

Scenario	Params	LR	Iters	Batch/GPU	GPUs	Eff. batch	Architecture hyperparameters
Cylinder flow	41.3 M	1×10^{-3}	4,000	4	4	16	`embed_dim`=1024, `depth`=6, `n_blocks`=8, `modes`=32, `patch_size`=8; warm-start `model_S.pth`
Controlled cylinder	41.3 M	5×10^{-4}	4,000	4	4	16	*Same architecture as Cylinder flow*
Turbulent combustion	41.3 M	1×10^{-3}	10,000	4	4	16	*Same architecture as Cylinder flow*

K.3 Fisher Spectrum across Training Stages

For the four architectures evaluated in the main text, the empirical Fisher Information Matrix concentrates its energy in dominant directions (Table C.1). This allows PhysGuard to protect a compact subspace while retaining an orthogonal complement for adaptation.

DPOT-S requires a larger fraction of Fisher directions to capture 90% of the Fisher mass at the checkpoint used for adaptation. We track the training stages to distinguish this observation from an attribution to joint pretraining or the DiT backbone.

We evaluate four stages of DPOT-S on Cylinder Flow with the same 200 simulation samples and layer-wise estimator. They are random initialization, the jointly pretrained checkpoint, and checkpoints after 80 and 4,000 steps of scenario-specific training. The last is the checkpoint used for adaptation. Table K.2 reports the fraction of eigendirections needed to capture 90% of Fisher mass, summarized by the median across layers and separately within DiT attention layers. A larger fraction indicates a flatter spectrum. Evenness measures how uniformly Fisher mass is distributed across the six DiT blocks, with 1.0 denoting uniformity.

Table K.2: DPOT-S Fisher-spectrum flatness across training stages on Cylinder Flow. Dashes indicate unreported values.

Training stage	Median fraction for 90% Fisher mass	Within DiT blocks	Evenness across DiT blocks
Random initialisation (control)	—	2.0%	0.88
After joint pretraining, before scenario training	5.5%	6.0%	0.81
After 80 steps of scenario training	5.5%	7.0%	0.87
After 4,000 steps (evaluated checkpoint)	13.0%	21.0%	0.97

The median fraction remains at 5.5% through the first 80 steps and rises to 13.0% at step 4,000. Within DiT blocks it rises from 6.0% after joint pretraining to 21.0%, while evenness reaches 0.97. Thus the pronounced flattening develops during scenario-specific training. These measurements identify when it appears, but do not isolate an architectural cause. The negative transfer results in Table K.3 are consistent with this checkpoint dependence; the spectrum should be assessed at the checkpoint used for adaptation.

K.4 Quantitative Results

Table K.3 reports DPOT-S performance across all three RealPDEBench scenarios using the same baselines and metrics as Table 1.

Table K.3: DPOT-S results on RealPDEBench. Blue and green mark the best and second-best values.

Architecture	Method	Cylinder Flow				Controlled Cylinder				Turbulent Combustion			
		RMSE ↓	R^2 ↑	fRMSE ↓	Low-f ↓	RMSE ↓	R^2 ↑	fRMSE ↓	Low-f ↓	RMSE ↓	R^2 ↑	fRMSE ↓	Low-f ↓
DPOT-S (41.3 M)	Pretrained	0.05663	0.8287	0.00838	0.01202	0.02187	0.9049	0.00333	0.00278	0.03957	0.1351	0.00549	0.00935
	DFT	0.04544	0.8897	0.00667	0.01026	**0.00996**	0.9803	0.00106	0.00089	**0.02668**	**0.6066**	**0.00350**	**0.00388**
	L_2-SP	0.04687	0.8827	0.00698	0.01075	0.01126	0.9748	0.00119	0.00098	0.02866	0.5463	0.00388	0.00484
	EWC	**0.04509**	**0.8914**	**0.00660**	0.00989	**0.00996**	**0.9803**	**0.00105**	**0.00088**	0.02723	0.5904	0.00361	0.00415
	PhysGuard	0.04893	0.8721	0.00690	**0.00935**	0.01350	0.9638	0.00138	0.00109	0.03712	0.2388	0.00493	0.00767

K.5 Qualitative Visualisations

Figures K.1–K.3 show DPOT-S predictions on the three scenarios under each baseline. They use the same layout as the per-architecture panels in Appendix J. Consistent with Table K.3, PhysGuard traces follow the ground truth closely on Cylinder Flow but underperform DFT/EWC on Controlled Cylinder and Turbulent Combustion, visible as slightly blurrier vortex boundaries and weaker local intensity.

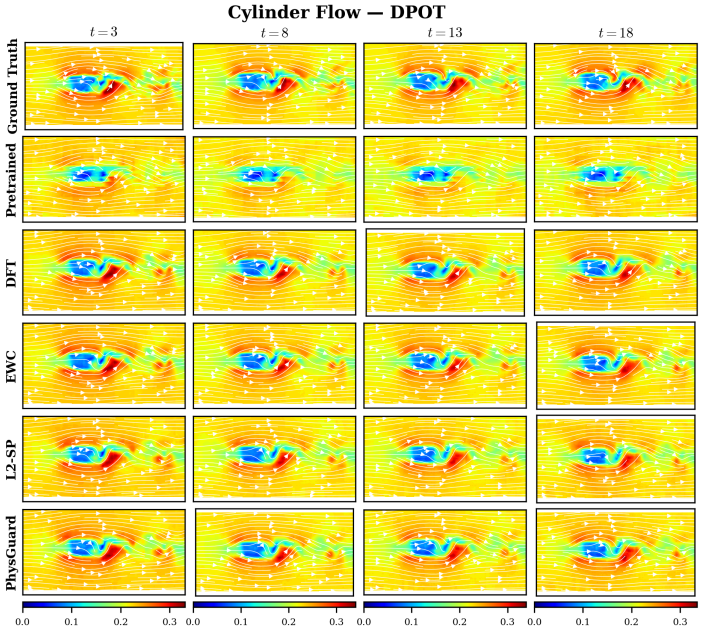


Figure K.1: DPOT-S predictions on Cylinder Flow.

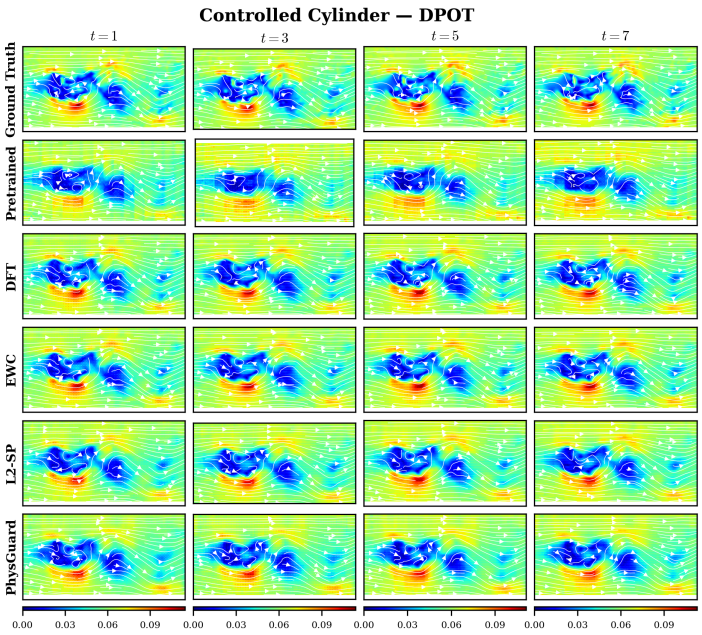


Figure K.2: DPOT-S predictions on Controlled Cylinder Flow.

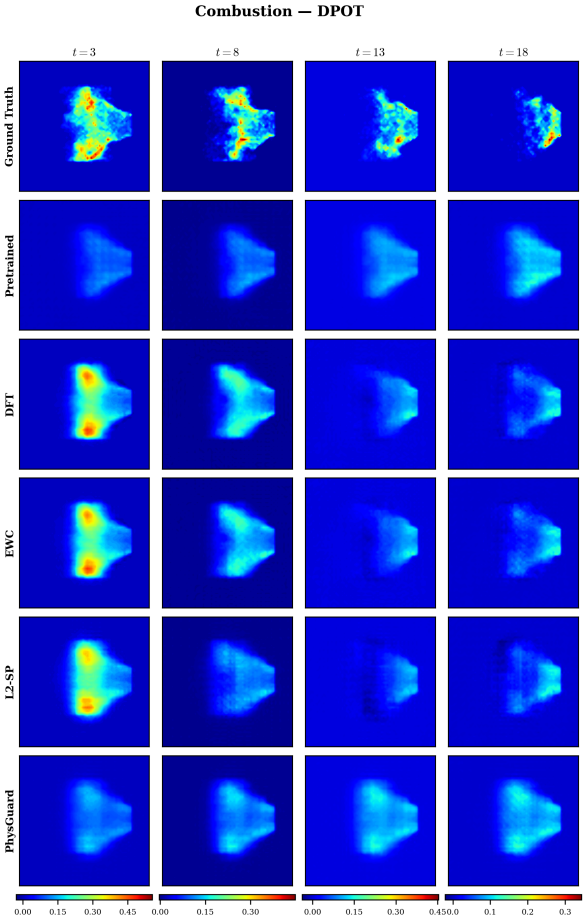


Figure K.3: DPOT-S predictions on Turbulent Combustion.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The abstract and Section 1 describe the contributions evaluated in Section 4.

2. **Limitations**

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

   Answer: [Yes]

   Justification: Section 5 discusses spectral assumptions and the limited benchmark scope.

3. **Theory assumptions and proofs**

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

   Answer: [Yes]

   Justification: Appendices D and E provide the Gram-matrix proof and complex-weight derivation.

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: Algorithm 1 and Appendix C describe the method and experimental protocol.

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

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

   Answer: [Yes]

   Justification: The code is open source, and RealPDEBench is publicly available.

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: Appendix C specifies data splits, architectures, optimizers and training schedules.

7. **Experiment statistical significance**

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

   Answer: [Yes]

   Justification: Appendix H.1 reports three-seed means and available standard deviations for DFT and PhysGuard.

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: Appendices C and D report hardware, memory, training budgets and Fisher estimation times.

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: The study uses public scientific data and involves no human subjects or private data collection.

10. **Broader impacts**

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

    Answer: [Yes]

    Justification: Appendix A discusses model reuse and risks from residual prediction errors.

11. **Safeguards**

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

    Answer: [N/A]

    Justification: No high-risk generative models or scraped datasets are introduced.

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: We credit RealPDEBench (CC BY-NC 4.0) and DPOT (Apache 2.0).

13. **New assets**

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

    Answer: [Yes]

    Justification: The repository README documents installation, training and evaluation.

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

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

    Answer: [N/A]

    Justification: No crowdsourcing or human subjects are involved.

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

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

    Answer: [N/A]

    Justification: No human subjects research is conducted.

16. **Declaration of LLM usage**

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

    Answer: [N/A]

    Justification: LLMs are used only for writing assistance.