PHYLOOPNO: LOOPED NEURAL OPERATOR WITH PHYSICS-GATED DENSE EXPERT LAYERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Low-rank Transformer neural operators solve partial differential equations (PDEs) by modeling global interactions through compact latent representations. Slicing introduces an explicit latent bottleneck; global interactions are then realized through latent-token attention followed by deslicing, or through deslicing decoupled from slicing. We observe that slicing and deslicing are the main sources of differentiation across layers. Assigning layer-specific parameters to most other components offers relatively limited improvements in model performance, resulting in low parameter efficiency. Inspired by looped Transformers in large language models, we introduce PHYLOOPNO, a parameter-efficient neural operator built on LinearNO. It reuses operator and feed-forward transformations across loop iterations while retaining independent slicing and deslicing, preserving spatial interactions that vary with depth. However, parameter sharing alone reduces parameter capacity and can limit predictive accuracy. To address this trade-off, we replace the pointwise feed-forward network with physics-gated dense experts. Inspired by domain decomposition, a learned gate softly groups points according to their physical features and combines the responses of all parallel experts at each point. Experiments show the best results on five of six standard PDE benchmarks in our comparison and competitive performance on industrial aerodynamic tasks. Increasing depth from 8 to 40 layers further reduces its errors on Airfoil, Darcy, and Elasticity overall. Selective parameter reuse lowers parameter growth on Airfoil and Darcy, enabling parameter-efficient depth scaling.

1 INTRODUCTION

Transformer-based neural operators have emerged as a promising approach to solving partial differential equations (PDEs) across complex geometries. Recent models, including Transolver and LinearNO, make global interactions tractable by organizing spatial information through compact latent representations (Wu et al., 2024; Hu et al., 2026). These low-rank architectures have substantially advanced efficient PDE learning. Their development has largely centered on improving spatial interactions, while the joint expansion of nonlinear capacity and computational depth under a limited parameter budget remains less explored. Developing neural operators that can scale effectively along these dimensions is an important step toward more capable PDE solvers.

Two natural scaling directions are to increase the number of slices, which enlarges the latent representation, and to stack more operator layers. Increasing the slice count raises projection and spatial-interaction costs, yet its accuracy gains can diminish or saturate, as observed in LinearNO's slice-number study (Hu et al., 2026). Increasing depth, meanwhile, typically introduces a separate set of parameters for every added layer, with gains that remain task dependent (Wu et al., 2024). These approaches couple model expansion to additional resources without ensuring that the added capacity is used effectively. The challenge is therefore to allocate capacity more effectively and make it reusable throughout the computation.

A PDE solver must coordinate two types of computation. Spatial aggregation communicates information between locations, whereas pointwise nonlinear transformations update the representation at each location. Increasing the size of the latent representation primarily changes the former; increasing the diversity of pointwise transformations changes the latter. Treating these components separately makes it possible to ask where additional parameters are most useful. In particular, spatial interactions

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

may need to change as the representation evolves, even when the same nonlinear transformations remain useful at several stages. This motivates a design that separates reusable transformations from the mechanisms that adapt their application to the current field.

Parameter reuse also changes how width and depth interact. With an independent layer at every depth, adding experts introduces a new expert bank at every stage. Repeating a shared bank allows its transformations to be applied to successively updated features, while an independent gate changes their relative contributions at each visit. The resulting model can devote parameters to a broader collection of nonlinear responses without replicating that collection throughout the network. Our experiments therefore examine both the benefit of adaptive expert mixing at fixed total hidden width and the behavior of the model as executed depth increases. These comparisons distinguish the role of the gate from that of parameter reuse, and relate prediction accuracy to the parameter cost of the chosen execution schedule.

In this work, we present PHYLOOPNO, a Transformer neural operator designed for parameter-efficient scaling in both nonlinear width and computational depth. Our central idea is to expand the transformations available to different physical states and reuse them across depth. Building on LinearNO, we introduce *physics-gated dense experts*, which combine parallel nonlinear experts according to the evolving physical features at each spatial point. This provides a width-scaling mechanism in the pointwise transformation, complementing expansion of the spatial latent representation. To make this capacity available at successive computational stages, we draw on looped Transformers in large language models (Bae et al., 2025; Jaggi, 2026) and repeatedly apply a shared operator core and expert banks. Lightweight components specific to each execution adapt spatial aggregation and expert mixing to the evolving field. Together, these designs expand the available nonlinear transformations and their repeated application while controlling parameter growth.

Our contributions are summarized as follows:

- We introduce physics-gated dense experts to scale the nonlinear width of neural operators. By combining parallel experts according to learned physical features, the model expands its state-dependent capacity without enlarging the spatial latent bottleneck.
- We develop a looped Transformer neural operator that reuses operator and expert parameters across successive computational stages. This design enables parameter-efficient depth expansion while allowing each execution to adapt spatial aggregation and expert mixing.

2 RELATED WORK

2.1 NEURAL OPERATORS

Neural operators learn mappings between function spaces to approximate families of PDE solutions (Kovachki et al., 2023). DeepONet combines branch and trunk networks to represent the dependence on input functions and query locations (Lu et al., 2021), while Fourier neural operators (FNO) parameterize integral kernels in the Fourier domain (Li et al., 2021). U-NO introduces a U-shaped architecture with skip connections to support deeper models with reduced memory requirements (Rahman et al., 2023); F-FNO uses separable spectral layers and revised residual connections to improve efficiency and depth scaling (Tran et al., 2023). For irregular geometries, Geo-FNO learns deformations to a regular latent domain (Li et al., 2023c), and GINO combines graph-based geometric mappings with Fourier operators (Li et al., 2023d). Latent Spectral Models (LSM) perform spectral operator learning in a learned latent space (Wu et al., 2023). These directions improve spectral representations, architectural efficiency, and geometric flexibility. Our work complements them by organizing nonlinear capacity and parameter reuse within a Transformer neural operator.

2.2 LATENT TOKENS AND BOTTLENECK ARCHITECTURES

Latent-token architectures route spatial information through a compact bottleneck to reduce the cost of global interactions. IPOT uses learned inducing points to separate its latent attention processor from the input and output discretizations (Lee & Oh, 2024). LNO maps geometric features into latent tokens through Physics-Cross-Attention, applies Transformer layers in that space, and decodes predictions at query locations (Wang & Wang, 2024). Universal Physics Transformers (UPT) similarly process

compressed latent representations and support latent-space evolution across different discretizations (Alkin et al., 2024). Another design maintains the evolving state on mesh points and introduces latent tokens within each operator layer. Transolver aggregates points into physical slices, applies attention to the resulting tokens, and reconstructs point features (Wu et al., 2024). LinearNO decouples slicing and deslicing and removes the intervening slice attention (Hu et al., 2026). PHYLOOPNO retains this slicing–deslicing backbone while expanding nonlinear width through physics-gated dense experts and reusing the operator core across depth.

The latent bottleneck and the pointwise transformation control different aspects of an operator layer. In Transolver and LinearNO, slicing aggregates point features into latent summaries, while deslicing returns information to the spatial domain (Wu et al., 2024; Hu et al., 2026). These operations define the paths through which distant points exchange information. A pointwise FFN instead transforms the features available at each point, without introducing a separate channel of spatial communication. Increasing its nonlinear capacity therefore complements changes to latent interactions. This distinction motivates treating spatial routing and pointwise transformation as separate design choices: our model retains LinearNO's latent bottleneck and expands the pointwise branch through feature-dependent mixtures of dense experts, without adding latent-token attention.

2.3 LOOPED TRANSFORMERS

Universal Transformers combine attention with recurrent depth and shared transformations (Dehghani et al., 2019). Subsequent work studies the algorithmic capabilities, parameter efficiency, and approximation behavior of looped architectures (Giannou et al., 2023; Yang et al., 2024; Xu & Sato, 2025). Relaxed Recursive Transformers introduce lightweight deviations from shared layers (Bae et al., 2025), while Tying the Loop investigates shared experts with untied interfaces in language models (Jaggi, 2026). We apply parameter reuse to PDE operators through visit-specific slicing, deslicing, normalization, and feature gates.

Looped architectures also separate executed depth from the number of independently stored transformations (Dehghani et al., 2019; Bae et al., 2025; Jaggi, 2026). For neural operators, this separation raises a further question: which components should be reused when the intermediate field changes across repeated executions? Sharing an entire layer reuses the parameters that generate spatial routing weights as well as its feature transformations. Our selective-sharing design instead reuses the input, value, and output transformations and the expert bank at each stored core position, while retaining visit-specific slicing, deslicing, normalization, and expert gates. Thus, repeated visits reuse nonlinear transformations but can aggregate spatial information and combine expert responses differently. Distinct core positions retain their own parameters, so reuse occurs across visits to the same position.

3 METHOD

3.1 PROBLEM SETUP

We learn a PDE solution operator $\mathcal{G}_\theta : a \mapsto u$, where a encodes physical inputs and geometry on a domain Ω, u is the solution field, and θ denotes trainable parameters. Inputs at N sampled points are lifted to features $X \in \mathbb{R}^{N\times C}$, with C channels. A readout maps the final features to predictions. Our backbone, LinearNO (Hu et al., 2026), uses independently learned slicing and deslicing maps. Within one attention head, slicing weights K pool projected point values V into M latent summaries S, and deslicing weights Q reconstruct point features Y:

$$S = K^\top V, \qquad Y = QS. \tag{1}$$

Here $K, Q \in \mathbb{R}^{N\times M}$ are normalized weights, and $\top$ denotes transpose. The latent transformation is the identity: the summaries pass directly from slicing to deslicing. This factorized linear attention scales linearly with the point count for fixed latent and channel widths.

PHYLOOPNO combines this spatial update with physics-gated dense experts, which mix the outputs of parallel feed-forward networks (FFNs) according to each point's features. Omitting the execution index, one layer updates the point features as

$$\begin{aligned} Z &= X + \mathcal{O}\big(\mathrm{LN}_1(X)\big), \\ X^+ &= Z + \alpha G\big(\mathrm{LN}_2(Z)\big). \end{aligned} \tag{2}$$

Here $\mathcal{O}$ is the LinearNO spatial operator, Z the features after the spatial update, and X^+ the layer output. The expert map G acts pointwise; $\mathrm{LN}_1, \mathrm{LN}_2$ are LayerNorm operations over channels, and α scales the expert update (Appendix C). Depth counts complete layer executions. Figure 1 summarizes PHYLOOPNO. Between independent prefix and suffix layers, the core repeats shared feature transformations and experts. Slicing, deslicing, gates, and normalization remain independent across visits.

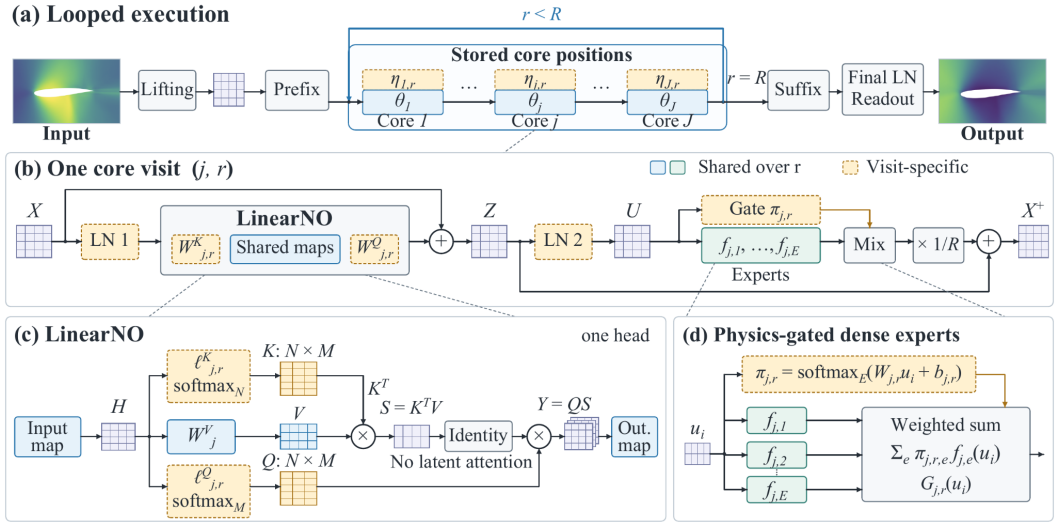

Figure 1: PHYLOOPNO architecture. (a) J stored core positions are repeated R times. (b) Each visit uses the two residual updates in Eq. (2), with $U = \mathrm{LN}_2(Z)$ and expert scaling $1/R$. (c) One-head slicing/deslicing forms $S = K^\top V$ and $Y = QS$ without latent attention. (d) All E experts process each point; a visit-specific gate mixes their outputs. Blue/green denote parameters shared across visits (θ_j); dashed gold denotes visit-specific parameters ($\eta_{j,r}$).

3.2 LOOPED NEURAL OPERATOR

To determine which transformations should remain independent under looping, Figure 2 compares three settings: *Baseline* is standard LinearNO; *Full-share* shares all core-layer parameters across repeated visits; and *Selective-share* retains independent slicing and deslicing. Both looped variants use a single pointwise FFN. The radial coordinate shows relative error reduction from Baseline: zero indicates equal performance, and negative values indicate higher error. Complete configurations and numerical results are given in Appendix C.5.

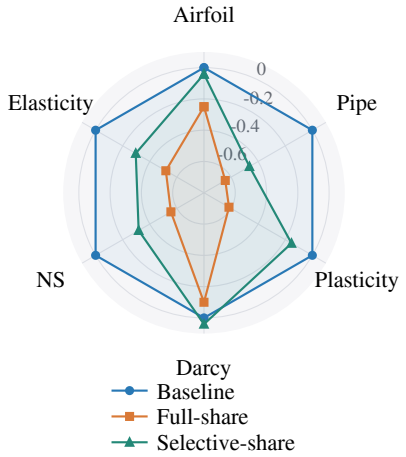

Figure 2: Relative error reduction from Baseline; higher is better. Selective-share improves over Full-share on all six tasks.

Fully sharing the core increases error on every benchmark, by up to 64.3% on Pipe (0.0028 to 0.0046). Retaining independent slicing and deslicing reduces all six errors. Airfoil approaches LinearNO (0.0050 versus 0.0048), while Darcy improves upon it (0.0048 versus 0.0050). These results support preserving depth-dependent spatial aggregation and reconstruction while reusing the remaining feature transformations. In Eq. (1), slicing determines which points form each latent summary, and deslicing determines how those summaries return to each point. Allowing these maps to change across visits provides a direct way to differentiate spatial interactions across depth.

Selective sharing and looped execution. Motivated by this comparison and looped Transformers in language models (Bae et al., 2025; Jaggi, 2026), we reuse an internal sequence of operator layers across loop iterations. Every eight-layer looped model uses the $1 + 3 \times 2 + 1$ schedule: one independent prefix layer, three core positions executed twice, and one independent suffix layer. The prefix processes the lifted input, and the suffix prepares features for the output readout. Parameter sharing is confined to the recurrent core between them. As shown in Figure 1(a), each of R iterations executes the J stored core positions in sequence and passes its output to the next iteration. Position j has shared parameters θ_j and visit-specific parameters $\eta_{j,r}$ for repetition r. Different positions have distinct parameters; repeated visits to one position reuse its input and value projections, output transformation, and pointwise FFN. Slicing and deslicing, including their effective temperatures, remain independent at every visit. This increases executed depth without storing a new copy of the shared transformations at every layer.

Sharing reduces the number of independently learned parameters. To expand nonlinear capacity within this budget, we replace the pointwise FFN with the physics-gated dense experts introduced in Section 3.3. Expert FFNs are shared across visits, while each visit has an independent affine router. These lightweight routers let successive layers assign different weights to the same experts, adding visit-specific pointwise responses at a small parameter cost. Figure 1(b) shows the two residual updates, with the spatial and expert branches detailed in panels (c) and (d). Normalization and complete parameter ownership are specified in Appendix C.

Spatial communication across repeated layers. Selective sharing preserves slicing and deslicing at every execution. The following result describes how a small change in input features affects a layer's output, separating the local response from the response carried by its latent summaries.

Theorem 1 (Latent-mediated differential structure)**.** *Fix the parameters, geometry, and sampled points of one* PHYLOOPNO *layer, including its residuals and expert update. Assume deterministic evaluation, differentiable component maps, and spatial mixing only through latent summaries and an optional local input projection. With M latent tokens per head and C total value channels, its input–output Jacobian J satisfies*

$$J = J_{\text{local}} + J_{\text{latent}}, \qquad \operatorname{rank} J_{\text{latent}} \le MC. \tag{3}$$

Here J maps small input changes to first-order output changes. The term J_{local} holds the summaries fixed and connects only points within the input projection's neighborhood; it acts independently at each point for a pointwise projection. The term J_{latent} transmits changes through the summaries. Its rank counts the independent output-response directions available through this path.

The result identifies the latent bottleneck through which slicing and deslicing communicate spatial changes. Both response components are reevaluated on the updated features at each visit. It applies to shared and unshared layers; the benefit of independent slicing and deslicing is supported by the ablation, rather than implied by the rank bound. Appendix B gives the complete proof and its extension to repeated execution.

3.3 PHYSICS-GATED DENSE EXPERTS

Physics gating. To increase nonlinear capacity under parameter sharing, we replace the pointwise FFN with parallel FFNs, called experts. Inspired by numerical domain decomposition (Toselli & Widlund, 2005), we learn soft assignments from physical features instead of prescribing fixed spatial subdomains. Let $z \in \mathbb{R}^C$ be one point's normalized features entering the expert branch in Eq. (2), and let E be the number of experts. Figure 1(d) denotes this vector by u_i, where i indexes spatial points. Omitting the core and visit indices, an affine gate produces expert scores $g \in \mathbb{R}^E$, which softmax converts into mixing weights:

$$g = W_g z + b_g, \qquad \pi_e(z) = \frac{\exp(g_e)}{\sum_{k=1}^{E} \exp(g_k)}. \tag{4}$$

Here $W_g \in \mathbb{R}^{E \times C}$ and $b_g \in \mathbb{R}^E$ are learned gate parameters; e identifies an expert and k runs over all experts. The vector $\pi(z)$ collects the positive weights $\pi_e(z)$, which sum to one. Points with similar features receive similar expert weights, allowing the pointwise transformation to adapt to different physical states. This is the physics-gating mechanism.

Dense expert mixing. Each expert has its own two-layer FFN with hidden width F:

$$f_e(z) = B_e \ \mathrm{GELU}(A_e z + a_e) + b_e. \tag{5}$$

The weight matrices A_e, B_e and biases a_e, b_e map C input features to F hidden features and then back to C output features; GELU applies a nonlinearity to each hidden feature. The expert outputs are combined as

$$G(z) = \sum_{e=1}^{E} \pi_e(z) f_e(z). \tag{6}$$

Every expert processes every point, contributing to $G(z) \in \mathbb{R}^C$ according to its gate weight. At each core position, expert parameters are shared across loop iterations, while independent gates allow different mixtures of the same experts. Sparse mixtures activate only a subset of experts and can suffer routing collapse, leaving rarely selected experts undertrained (Shazeer et al., 2017). When training on a single physical system, the risk of routing collapse is very high; we therefore use dense mixing. The expert bank contains EF hidden units in total. The following proposition relates one execution of this layer to a single FFN of the same total hidden width and identifies the response introduced by its gate.

Proposition 1 (Gated expert representation)**.** *At a fixed visit, let $E, C, F \geq 1$ be integers and let G be the mixture in Eq. (6), with E two-layer GELU experts $f_e : \mathbb{R}^C \to \mathbb{R}^C$ of hidden width F. Assume arbitrary finite real parameters in all affine layers and deterministic evaluation with dropout disabled. For any fixed positive mixing weights that sum to one, the functions represented by G, as the expert parameters vary, are exactly those of a two-layer GELU FFN of hidden width EF. Allowing the affine-softmax gate in Eq. (4) retains this entire function class. Moreover, at every $z \in \mathbb{R}^C$,*

$$J_G(z) = \sum_{e=1}^{E} \pi_e(z) J_{f_e}(z) + \underbrace{\sum_{e=1}^{E} f_e(z) \nabla \pi_e(z)^\top}_{J_{\mathrm{gate}}(z)}. \tag{7}$$

Here J_G and J_{f_e} are the Jacobian matrices of the mixture and expert outputs with respect to z, describing their first-order sensitivity. The column gradient $\nabla \pi_e$ describes how expert e's weight changes with that input. The gating term $J_{\mathrm{gate}}(z)$ has rank at most $\min(C, E-1)$.

Proof sketch. For constant mixing weights, concatenating the experts' hidden units and absorbing their weights into the output matrices gives a width-EF FFN; partitioning such an FFN into E groups gives the converse. The affine gate can realize these constant weights. The product rule gives Eq. (7). The softmax derivative with respect to its E scores has rank $E-1$, so the gating term has rank at most $\min(C, E-1)$. The complete proof is in Appendix A. □

The first term describes changes in the expert outputs, while the second describes changes in their relative contributions. The proposition guarantees inclusion of the width-EF FFN function class at one visit; it does not guarantee recovery of an entire unshared network's capacity or lower prediction error. Section 4.2 evaluates the benefit of adaptive mixing against a single FFN of the same total hidden width.

4 Experiments

4.1 Main Results

Baselines. We compare with spectral and latent-space operators, attention-based operators, and recent Transformer solvers listed in Table 1. LinearNO serves as the backbone baseline. Industrial comparisons additionally include graph-based and geometry-aware models listed in Table 2. **Benchmarks.** We evaluate on six standard PDE benchmarks—Airfoil, Pipe, Plasticity, Darcy, Navier–Stokes (NS), and Elasticity—using test relative L_2 errors (Table 1). ShapeNet-Car and AirfRANS assess field prediction and aerodynamic-coefficient accuracy on variable geometries and larger spatial discretizations (Table 2). Dataset descriptions and training settings are given in Appendix C.

PhyLoopNO achieves the lowest error on five of the six benchmarks in Table 1: Airfoil, Plasticity, Darcy, NS, and Elasticity. Relative to our reproduced LinearNO, error decreases by 4.2%, 7.7%, 4.0%,

5.5%, and 13.0%, respectively. Transolver++ performs best on Pipe. Our LinearNO reproduction follows the official implementation and settings. Compared with the published errors of 0.0024 on Pipe and 0.0050 on Elasticity, errors increase by 16.7% and 8.0%, respectively (Hu et al., 2026).

Table 1: Test relative L_2 errors on six standard PDE benchmarks (lower is better). Best results are **bold** and second-best results are underlined, including ties. NS denotes Navier–Stokes; / indicates an unavailable result. † marks our reproduction.

Model	Structured Mesh			Regular Grid		Point Cloud
	Airfoil	Pipe	Plasticity	Darcy	NS	Elasticity
FNO (2021)	/	/	/	0.0108	0.1556	/
WMT (2021)	0.0075	0.0077	0.0076	0.0082	0.1541	0.0359
U-FNO (2022)	0.0269	0.0056	0.0039	0.0183	0.2231	0.0239
Geo-FNO (2023c)	0.0138	0.0067	0.0074	0.0108	0.1556	0.0229
U-NO (2023)	0.0078	0.0100	0.0034	0.0113	0.1713	0.0258
F-FNO (2023)	0.0078	0.0070	0.0047	0.0077	0.2322	0.0263
LSM (2023)	0.0059	0.0050	0.0025	0.0065	0.1535	0.0218
Galerkin (2021)	0.0118	0.0098	0.0120	0.0084	0.1401	0.0240
HT-Net (2022)	0.0065	0.0059	0.0333	0.0079	0.1847	/
OFormer (2023a)	0.0183	0.0168	0.0017	0.0124	0.1705	0.0183
FactFormer (2023b)	0.0071	0.0060	0.0312	0.0109	0.1214	/
ONO (2024)	0.0061	0.0052	0.0048	0.0076	0.1195	0.0118
Transolver (2024)	0.0053	0.0030	0.0013	0.0055	0.0882	0.0065
Transolver++ (2025)	0.0051	**0.0027**	0.0014	0.0056	0.1010	0.0064
LinearNO† (2026)	0.0048	0.0028	0.0013	0.0050	0.0752	0.0054
PHYLOOPNO (ours)	**0.0046**	0.0036	**0.0012**	**0.0048**	**0.0711**	**0.0047**

We search $E \in \{1, 2, 3, 4\}$ on all six standard benchmarks and select four experts for Airfoil, Plasticity, NS, and Elasticity, three for Pipe, and one for Darcy. The four-expert NS model achieves an error of 0.0711, outperforming the reproduced LinearNO baseline. More experts do not consistently improve accuracy. Increasing to eight experts raises the test relative L_2 error on most evaluated datasets, with particularly marked degradation on Elasticity, where the error reaches 0.0071. Task configurations are provided in Appendix C.

Field predictions. Figure 3 visualizes predictions from the configurations in Table 1 on Airfoil, Darcy, Elasticity, and Pipe. The predicted fields reproduce the main spatial patterns across different geometries and discretizations. The absolute error maps reveal localized discrepancies, including the sharp transition near the airfoil and portions of the hole boundary in Elasticity.

On AirfRANS and ShapeNet-Car, we evaluate PHYLOOPNO and reproduce LinearNO following its official experimental protocol. Table 2 shows lower volume and surface errors for PHYLOOPNO on both datasets, with competitive lift and drag prediction accuracy. **Parameter efficiency.** In a separate three-expert Airfoil configuration, PHYLOOPNO uses 17.46% fewer trainable parameters than LinearNO at the same executed depth. Reusing the backbone and expert FFNs accommodates additional expert capacity within a smaller parameter budget. The comparison includes all shared and visit-specific parameters; the configuration and accounting are given in Appendix C.3.

4.2 ABLATION STUDIES

Slicing/deslicing sharing. We test reusing the same slicing and deslicing parameters whenever a core layer is repeated, including their effective temperatures. Slicing and deslicing remain distinct maps, while routers and normalization remain independent between loop iterations. Table 3 compares this variant with the full model, using the task-specific expert counts selected for Table 1. Independent slicing and deslicing reduce the error on all four tasks. Sharing these projections increases the error from 0.0047 to 0.0064 on Elasticity and from 0.0036 to 0.0044 on Pipe, indicating a stronger dependence on independently learned spatial transformations in these tasks.

Depth scaling. We evaluate depths of 8, 16, 24, 32, and 40 layers, keeping channel width, latent count, and task-specific expert count fixed. Figure 4 compares the three models on Airfoil, Darcy,

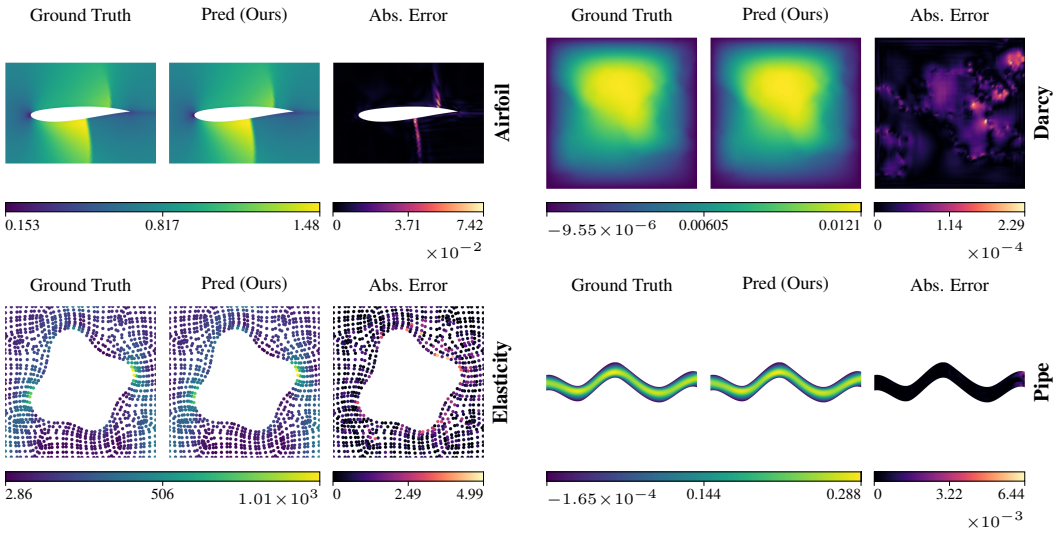

Figure 3: Field predictions for the configurations in Table 1. Top: Airfoil (left) and Darcy (right); bottom: Elasticity (left) and Pipe (right). Each group shows the reference field, PHYLOOPNO prediction, and pointwise absolute error $|\widehat{u} - u|$. Reference and prediction share a color scale within each dataset; errors use a separate scale. The fields are Mach number, Darcy solution, stress, and velocity, respectively.

Table 2: Industrial aerodynamic results. Volume and surface columns report relative L_2 errors: pressure for AirfRANS, surrounding velocity and surface pressure for ShapeNet-Car. C_L and C_D denote lift and drag coefficients; their columns report relative errors. ρ_L and ρ_D are Spearman rank correlations. † marks our reproduced LinearNO baseline; other baseline values follow Hu et al. (2026), including their reproductions of LNO and Transolver. Arrows indicate the preferred direction. Best results are **bold** and second-best results are underlined, including ties.

	AirfRANS				ShapeNet-Car			
Model	Vol.↓	Surf.↓	C_L ↓	ρ_L ↑	Vol.↓	Surf.↓	C_D ↓	ρ_D ↑
GraphSAGE (2017)	0.0087	0.0184	0.1476	0.9964	0.0461	0.1050	0.0270	0.9695
Graph U-Net (2019)	0.0076	0.0144	0.1677	0.9949	0.0471	0.1102	0.0226	0.9725
MeshGraphNet (2021)	0.0214	0.0387	0.2252	0.9945	0.0354	0.0781	0.0168	0.9840
GNO (2020)	0.0269	0.0405	0.2016	0.9938	0.0383	0.0815	0.0172	0.9834
Geo-FNO (2023c)	0.0361	0.0301	0.6161	0.9257	0.1670	0.2378	0.0664	0.8280
GINO (2023d)	0.0297	0.0482	0.1821	0.9958	0.0386	0.0810	0.0184	0.9826
LNO (2024)	0.0214	0.0268	0.1480	0.9744	0.0269	0.0870	0.0174	0.9781
Transolver (2024)	0.0023	0.0085	0.1230	**0.9978**	0.0221	0.0785	0.0117	0.9914
LinearNO† (2026)	0.0018	0.0078	**0.0481**	0.9918	0.0195	0.0752	**0.0108**	**0.9924**
PHYLOOPNO (ours)	**0.0015**	**0.0076**	0.0488	0.9911	**0.0188**	**0.0744**	**0.0108**	0.9910

Table 3: Effect of sharing slicing/deslicing parameters between loop iterations, measured by test relative L_2 error. Both variants use the same task-specific expert counts as Table 1.

Configuration	Airfoil	Darcy	Elasticity	Pipe
Full model	0.0046	0.0048	0.0047	0.0036
Shared slicing/deslicing	0.0052	0.0051	0.0064	0.0044

and Elasticity; numerical results appear in Appendix C.4. From 8 to 40 layers, PHYLOOPNO reduces error by 30.4% on Airfoil, 29.2% on Darcy, and 12.8% on Elasticity. Airfoil and Darcy improve at every measured depth; Elasticity improves overall after a small increase at 16 layers. At 40 layers, the respective errors are 0.0032, 0.0034, and 0.0041. Among the three models, PHYLOOPNO gives the lowest errors on Airfoil and Elasticity and matches LinearNO on Darcy.

Selective sharing makes the added depth less expensive in parameters on Airfoil and Darcy. Under the two-repeat scaling schedule in Appendix C.3, parameter accounting gives the largest savings on Darcy: expanding from 8 to 40 layers adds 66.48% fewer parameters than Transolver and 46.26% fewer than

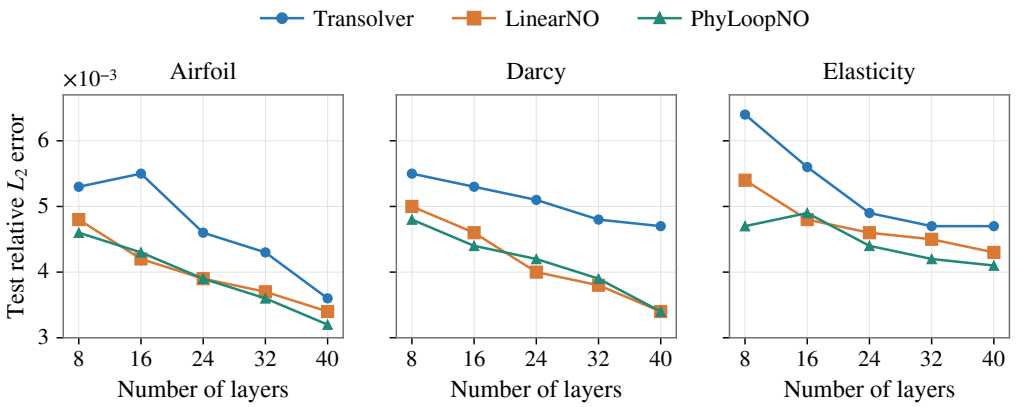

Figure 4: Depth scaling on Airfoil (left), Darcy (middle), and Elasticity (right). Markers indicate test relative L_2 errors at each evaluated depth; lower is better.

LinearNO. This comparison uses one expert on Darcy and includes independent slicing/deslicing, routers, and normalization. The reduction concerns the added parameters, rather than the total model size. On Plasticity, all three models show little further improvement by 40 layers, and PHYLOOPNO offers no clear error advantage in this saturated regime.

Physics-gated experts. For each dataset in Table 1, we replace the physics-gated dense experts with a single two-layer FFN of hidden width EF, equal to the combined hidden width of the E experts in that dataset's configuration. All other model and training settings remain unchanged. Test relative L_2 error increases on every dataset except Darcy, where it remains 0.0048. Darcy already uses one expert, for which the gate is identically one, so the replacement yields the same selective-share single-FFN model. Elasticity shows the largest increase, from 0.0047 to 0.0054, a relative increase of 14.9%. These results support the benefit of adaptive expert mixing at the same total hidden width.

5 CONCLUSION

We introduced PHYLOOPNO, a looped linear neural operator with physics-gated dense experts. The gate adapts the pointwise nonlinear transformation to the evolving physical representation, while parameter reuse distributes an expert bank and operator backbone across multiple executions. Independent slicing, deslicing, routing, and normalization preserve visit-specific computation. Our analysis connects the expert layer to wide FFNs and identifies the latent-mediated structure of the layer's differential response. The model achieves the best results on five of six standard PDE benchmarks in our comparison, while the ablations examine adaptive expert mixing and depth scaling.

The architecture opens a direction for bringing looped Transformers into scientific machine learning and for constructing parameter-efficient PDE foundation models. Physics-gated dense experts offer a complementary way to expand the capacity of PDE Transformers through state-dependent combinations of reusable transformations. Extending this looped design to Transolver and other attention-based neural operators, and studying shared models across multiple physical systems, are natural next steps.

REPRODUCIBILITY STATEMENT

Section 3 specifies the complete layer computation and parameter-sharing pattern. Appendix C gives tensor shapes, initialization, execution schedules, task configurations, and evaluation conventions. Theorem 1 and Proposition 1 state their assumptions in the main text; their complete proofs are given in Appendices B and A, respectively.

AI USE STATEMENT

Generative AI tools were used for literature exploration, model-design discussion and implementation planning, manuscript drafting, mathematical derivation and checking, and preparation of LaTeX and the architecture diagram. They were not used to generate experimental measurements or solution-field visualizations. Responsibility for the scientific claims, evidence, and final manuscript rests with the authors.

REFERENCES

Benedikt Alkin, Andreas Fürst, Simon Schmid, Lukas Gruber, Markus Holzleitner, and Johannes Brandstetter. Universal physics transformers: A framework for efficiently scaling neural operators. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://papers.nips.cc/paper_files/paper/2024/hash/2cd36d327f33d47b372d4711edd08de0-Abstract-Conference.html.

Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. Relaxed recursive transformers: Effective parameter sharing with layer-wise LoRA. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/54d6a55225cebbdc16fbb0e45c5bdf2b-Abstract-Conference.html.

Shuhao Cao. Choose a transformer: Fourier or Galerkin. In *Advances in Neural Information Processing Systems*, volume 34, 2021. URL https://proceedings.nips.cc/paper_files/paper/2021/hash/d0921d442ee91b896ad95059d13df618-Abstract.html.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL https://research.google/pubs/universal-transformers/.

Hongyang Gao and Shuiwang Ji. Graph u-nets. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pp. 2083–2092. PMLR, 2019. URL https://proceedings.mlr.press/v97/gao19a.html.

Angeliki Giannou, Shashank Rajput, Jy-Yong Sohn, Kangwook Lee, Jason D. Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pp. 11398–11442. PMLR, 2023. URL https://proceedings.mlr.press/v202/giannou23a.html.

Gaurav Gupta, Xiongye Xiao, and Paul Bogdan. Multiwavelet-based operator learning for differential equations. In *Advances in Neural Information Processing Systems*, volume 34, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/c9e5c2b59d98488fe1070e744041ea0e-Abstract.html.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, volume 30, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/5dd9db5e033da9c6fb5ba83c7a7ebea9-Abstract.html.

Wenjie Hu, Sidun Liu, Peng Qiao, Zhenglun Sun, and Yong Dou. Transolver is a linear transformer: Revisiting physics-attention through the lens of linear attention. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(1):408–416, 2026. doi: 10.1609/aaai.v40i1.37003. URL https://ojs.aaai.org/index.php/AAAI/article/view/37003.

Martin Jaggi. Tying the loop—tied expert layers in mixture-of-experts language models. *arXiv preprint arXiv:2606.16825*, 2026. URL https://arxiv.org/abs/2606.16825.

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. URL https://jmlr.org/papers/v24/21-1524.html.

Seungjun Lee and Taeil Oh. Inducing point operator transformer: A flexible and scalable architecture for solving PDEs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(1):153–161, 2024. doi: 10.1609/aaai.v38i1.27766. URL https://ojs.aaai.org/index.php/AAAI/article/view/27766.

Zijie Li, Kazem Meidani, and Amir Barati Farimani. Transformer for partial differential equations' operator learning. *Transactions on Machine Learning Research*, 2023a. URL https://openreview.net/forum?id=EPPqt3uERT.

Zijie Li, Dule Shu, and Amir Barati Farimani. Scalable transformer for PDE surrogate modeling. In *Advances in Neural Information Processing Systems*, volume 36, 2023b. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/590daf74f99ee85df3d8c007df9c8187-Abstract-Conference.html.

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

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=c8P9NQVtmnO.

Zongyi Li, Daniel Zhengyu Huang, Burigede Liu, and Anima Anandkumar. Fourier neural operator with learned deformations for PDEs on general geometries. *Journal of Machine Learning Research*, 24(388):1–26, 2023c. URL https://jmlr.org/papers/v24/23-0064.html.

Zongyi Li, Nikola Kovachki, Chris Choy, Boyi Li, Jean Kossaifi, Shourya Otta, Mohammad Amin Nabian, Maximilian Stadler, Christian Hundt, Kamyar Azizzadenesheli, and Animashree Anandkumar. Geometry-informed neural operator for large-scale 3D PDEs. In *Advances in Neural Information Processing Systems*, volume 36, pp. 35836–35854, 2023d. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/70518ea42831f02afc3a2828993935ad-Abstract-Conference.html.

Xinliang Liu, Bo Xu, and Lei Zhang. HT-Net: Hierarchical transformer based operator learning model for multiscale PDEs. *arXiv preprint arXiv:2210.10890v1*, 2022. URL https://arxiv.org/abs/2210.10890v1.

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*, 3:218–229, 2021. doi: 10.1038/s42256-021-00302-5. URL https://www.nature.com/articles/s42256-021-00302-5.

Huakun Luo, Haixu Wu, Hang Zhou, Lanxiang Xing, Yichen Di, Jianmin Wang, and Mingsheng Long. Transolver++: An accurate neural solver for PDEs on million-scale geometries. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 41432–41449. PMLR, 2025. URL https://proceedings.mlr.press/v267/luo25o.html.

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. URL https://openreview.net/forum?id=roNqYL0_XP.

Md Ashiqur Rahman, Zachary E. Ross, and Kamyar Azizzadenesheli. U-NO: U-shaped neural operators. *Transactions on Machine Learning Research*, 2023. URL https://openreview.net/forum?id=j3oQF9coJd.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=B1ckMDqlg.

Andrea Toselli and Olof B. Widlund. *Domain Decomposition Methods—Algorithms and Theory*, volume 34 of *Springer Series in Computational Mathematics*. Springer, 2005. doi: 10.1007/b137868.

Alasdair Tran, Alexander Mathews, Lexing Xie, and Cheng Soon Ong. Factorized fourier neural operators. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=tmIiMPl4IPa.

Tian Wang and Chuang Wang. Latent neural operator for solving forward and inverse PDE problems. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/39f6d5c2e310a5a629dcfc4d517aa0d1-Abstract-Conference.html.

Gege Wen, Zongyi Li, Kamyar Azizzadenesheli, Anima Anandkumar, and Sally M. Benson. U-FNO: An enhanced Fourier neural operator-based deep-learning model for multiphase flow. *Advances in Water Resources*, 2022. doi: 10.1016/j.advwatres.2022.104180. URL https://doi.org/10.1016/j.advwatres.2022.104180.

Haixu Wu, Tengge Hu, Huakun Luo, Jianmin Wang, and Mingsheng Long. Solving high-dimensional PDEs with latent spectral models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pp. 37417–37438. PMLR, 2023. URL https://proceedings.mlr.press/v202/wu23f.html.

Haixu Wu, Huakun Luo, Haowen Wang, Jianmin Wang, and Mingsheng Long. Transolver: A fast transformer solver for PDEs on general geometries. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 53681–53705. PMLR, 2024. URL https://proceedings.mlr.press/v235/wu24r.html.

Zipeng Xiao, Zhongkai Hao, Bokai Lin, Zhijie Deng, and Hang Su. Improved operator learning by orthogonal attention. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 54288–54299. PMLR, 2024. URL https://proceedings.mlr.press/v235/xiao24c.html.

Kevin Xu and Issei Sato. On expressive power of looped transformers: Theoretical analysis and enhancement via timestep encoding. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 69613–69646. PMLR, 2025. URL https://proceedings.mlr.press/v267/xu25x.html.

Liu Yang, Kangwook Lee, Robert Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/b8402301e7f06bdc97a31bfaa653dc32-Abstract-Conference.html.

A Proof of Proposition 1

At one visit, the expert layer in Section 3.3 is

$$G(z) = \sum_{e=1}^{E} \pi_e(z) f_e(z), \qquad f_e(z) = B_e \sigma(A_e z + a_e) + b_e, \qquad \pi(z) = \mathrm{softmax}(W_g z + b_g).$$

Here $z \in \mathbb{R}^C$ is the input feature, E is the expert count, F is each expert's hidden width, and σ is coordinatewise GELU. The expert parameters are $A_e \in \mathbb{R}^{F\times C}$, $a_e \in \mathbb{R}^F$, $B_e \in \mathbb{R}^{C\times F}$, and $b_e \in \mathbb{R}^C$; the gate parameters are $W_g \in \mathbb{R}^{E\times C}$ and $b_g \in \mathbb{R}^E$. All parameters are unrestricted finite real values, and dropout is disabled.

Wide-FFN representation. Stack the experts' first-layer parameters into $A \in \mathbb{R}^{EF\times C}$ and $a \in \mathbb{R}^{EF}$, and concatenate their output matrices into $B = [B_1 \ \cdots \ B_E]$. Then

$$G(z) = B \ \mathrm{diag}\big(\pi_1(z) I_F, \ldots, \pi_E(z) I_F\big) \sigma(Az + a) + \sum_{e=1}^{E} \pi_e(z) b_e. \tag{8}$$

Here I_F is the $F \times F$ identity and diag forms a block-diagonal matrix. For fixed positive weights $\bar{\pi}_e$ summing to one, absorb these weights into the output matrices and combine the output biases to obtain a width-EF FFN. Conversely, partition any width-EF FFN into E groups of F hidden units. Assign one group to each expert, divide its output matrix by $\bar{\pi}_e$, and give every expert the original output bias. Their weighted sum recovers the FFN. Finally, $W_g = 0$ and $(b_g)_e = \log \bar{\pi}_e$ realize the fixed weights, so adaptive gating retains this entire function class.

Differential response. Write J for the Jacobian with respect to z and ∇ for a column gradient. Collect expert outputs in $V_f(z) = [f_1(z) \ \cdots \ f_E(z)] \in \mathbb{R}^{C\times E}$. The product rule and softmax derivative give

$$J_G(z) = \sum_{e=1}^{E} \pi_e(z) J_{f_e}(z) + V_f(z)\big[\mathrm{diag}(\pi(z)) - \pi(z)\pi(z)^\top\big] W_g, \tag{9}$$

where $\mathrm{diag}(\pi)$ places the gate weights on the diagonal. The second term is J_{gate} in Eq. (7). The bracketed softmax derivative has quadratic form equal to the π-weighted variance of its argument's coordinates. Since every softmax weight is positive, its nullspace consists exactly of constant vectors, giving rank $E - 1$. Multiplication by V_f and W_g cannot increase rank, proving $\mathrm{rank}\, J_{\mathrm{gate}} \le \min(C, E - 1)$. □

The gating term vanishes when the gate is locally constant or all expert outputs coincide. It describes an additional response from input-dependent mixing, without establishing strict separation from every wider network class. Independent LayerNorm scales and shifts can change the inputs to shared experts at each visit, while independent gates can change their output contributions. Together they allow visit-dependent behavior without providing arbitrary independent expert matrices.

B Proof of Theorem 1

B.1 Layer Response

Fix the parameters, geometry, and N sampled points, and vectorize the input and output features as $x, y \in \mathbb{R}^{NC}$. Their Jacobian $J \in \mathbb{R}^{NC\times NC}$ differentiates feature values, not spatial coordinates. Collect all headwise summaries $S = K^\top V$ from Eq. (1) into $s(x) \in \mathbb{R}^{MC}$. Indeed, n_h heads with M tokens and $d_h = C/n_h$ value channels each give $n_h M d_h = MC$ entries. The map s includes slicing weights, softmax normalization, and projected values.

Regard the summaries temporarily as an independent input and write the complete layer as

$$y = \Phi\big(x, s(x)\big). \tag{10}$$

The map Φ computes deslicing weights, reconstruction, output processing, residuals, normalization, and the expert update. With summaries fixed, each output depends only on its input-projection

neighborhood: one point for pointwise projections or the convolutional stencil for local projections. Deslicing normalizes across tokens, and the subsequent transformations act separately at each point, introducing no further spatial aggregation.

Affine maps, local convolutions, GELU, softmax with positive temperatures, and channelwise LayerNorm with a positive stabilizer are differentiable. We disable dropout and hold temperatures and all other parameters fixed. The chain rule therefore yields

$$J = \underbrace{\frac{\partial \Phi}{\partial x}}_{J_{\text{local}}} + \underbrace{\frac{\partial \Phi}{\partial s} J_s}_{J_{\text{latent}}}, \qquad J_s = \frac{\partial s}{\partial x}. \tag{11}$$

The first term holds summaries fixed and connects only inputs within the projection neighborhood. The matrices $J_s \in \mathbb{R}^{MC \times NC}$ and $\partial \Phi / \partial s \in \mathbb{R}^{NC \times MC}$ map input changes to summary changes and then to output changes. Their product has rank at most MC, proving the theorem. □

All derivatives are evaluated at the current features and include changes in slicing/deslicing weights, expert outputs, and gate weights. The local term's coefficients may depend on global summaries, and the latent term may affect nearby as well as distant points. Thus, this decomposition separates computational paths, not strictly diagonal and off-diagonal matrix entries.

B.2 Repeated Execution

For T executed layers with common M and C, let $J_\ell = L_\ell + R_\ell$ be the Jacobian of execution ℓ at its incoming features, where L_ℓ and R_ℓ denote its local and latent terms. Thus $\operatorname{rank} R_\ell \le MC$. Products act from right to left; empty products are identities. Successively inserting $J_\ell - L_\ell = R_\ell$ gives the telescoping identity

$$J_T \cdots J_1 - L_T \cdots L_1 = \sum_{\ell=1}^{T} J_T \cdots J_{\ell+1} R_\ell L_{\ell-1} \cdots L_1. \tag{12}$$

Each summand has rank at most MC. Subadditivity of rank therefore gives

$$\operatorname{rank}\big(J_T \cdots J_1 - L_T \cdots L_1\big) \le TMC. \tag{13}$$

The local-path product remains pointwise for pointwise projections; convolutional receptive fields instead grow with depth and may span the domain. The bound counts executions and applies to both shared and independent layers. Shared nonlinear layers can have different Jacobians at different inputs, but the bound does not imply increasing rank or improved prediction accuracy.

C Implementation Details and Experimental Configurations

C.1 Layer Computation and Parameter Sharing

At a fixed core position, execution r in Eq. (2) uses the spatial operator $\mathcal{O}_r$, expert map G_r, and independent normalizations $\mathrm{LN}_{1,r}, \mathrm{LN}_{2,r}$. Write $U = \mathrm{LN}_{2,r}(Z)$ for the normalized features entering the experts; z in Section 3.3 is the feature vector of one point in U. For R core repetitions, $\alpha = 1/R$ in the core and $\alpha = 1$ in the independent prefix and suffix layers; the spatial residual has unit weight. Final normalization and readout are applied once.

In Eq. (1), each head uses $K, Q \in \mathbb{R}^{N \times M}$, values $V \in \mathbb{R}^{N \times d_h}$, and summaries $S \in \mathbb{R}^{M \times d_h}$, where n_h is the number of heads and $d_h = C/n_h$ is the channel width per head. In Figure 1(c), H denotes the features after the input map, and $\ell^K_{j,r}, \ell^Q_{j,r}$ are the slicing and deslicing logits, including their effective temperatures. The operations $\mathrm{softmax}_N$ and $\mathrm{softmax}_M$ normalize over points and latent tokens, respectively. Their bias-free $d_h \times M$ score projections are common to all heads; the value projection is $d_h \times d_h$. The affine router maps C channels to E expert scores. Its parameters $W_{j,r}, b_{j,r}$ in Figure 1(d) are the visit-specific instances of W_g, b_g in Eq. (4). Darcy, Airfoil, Pipe, and Plasticity retain 3×3 input convolutions; other tasks use pointwise projections. The convolutional variants and ShapeNet-Car use a Linear–GELU–Linear output transformation, and the others use a single projection.

Table 4: Parameter ownership at one repeated core position. Different positions always own different parameters.

Component	Across visits
Input projection (linear or native convolution)	Shared
Value projection W_V	Shared
Complete native output transformation	Shared
All dense expert weights and biases	Shared
Slicing and deslicing projections W_K, W_Q	Independent
Effective slicing/deslicing temperatures	Independent
Router matrix W_g and bias b_g	Independent
LayerNorm channel scales and shifts	Independent

Table 4 specifies sharing between executions of the same core position; different positions have distinct parameters. Prefix/suffix layers are independent and executed once. Independent slicing/deslicing tensors start from copied values without sharing storage. Routers start at zero, giving uniform weights; experts are initialized independently. Each LayerNorm starts with unit scale and zero shift. Shared parameters accumulate gradients from all executions.

C.2 Training, Data, and Evaluation

All tasks retain the native LinearNO input processing, output heads, losses, and optimization protocols. Tables 5 and 6 summarize configurations and data splits. Standard tasks use eight heads and AdamW with initial learning rate 10^{-3} for 500 epochs. Each expert retains the original FFN width, and looping leaves the latent count unchanged. Expert selection follows Section 4.1. AirfRANS and ShapeNet-Car use $C = 256$, $M = 32$, $F = 512$, and eight heads, with Adam, learning rate 10^{-3}, and batch size 1. They train for 400 and 200 epochs, respectively. NS predicts ten future frames autoregressively from ten observed frames; Plasticity uses twenty temporal queries and four output channels.

Table 5: Standard-benchmark configurations. C, M, F, and E denote channel width, latent tokens per head, expert hidden width, and expert count.

Task	C	M	F	E	Batch size	Scheduler
Airfoil	128	64	128	4	4	OneCycle
Pipe	128	64	128	3	4	OneCycle
Plasticity	128	64	128	4	8	OneCycle
Darcy	128	64	128	1	4	OneCycle
Navier–Stokes	256	32	512	4	2	OneCycle
Elasticity	128	64	128	4	1	Cosine

Table 6: Data splits and prediction targets. AirfRANS additionally uses 80 validation cases and 32,000 sampled training points, with full-graph evaluation. ShapeNet-Car uses fold 0 and variable point counts.

Task	Train	Test	Spatial representation	Target
Elasticity	1000	200	972-point cloud	Stress
Plasticity	900	80	101×31 grid	Deformation
Darcy	1000	200	85×85 grid	Pressure
Pipe	1000	200	129×129 mesh	Velocity
Airfoil	1000	200	221×51 mesh	Mach number
Navier–Stokes	1000	200	64×64 grid	Future vorticity
AirfRANS	720	200	Variable mesh	Flow variables
ShapeNet-Car	789	100	Variable mesh	Velocity, pressure

For n_{test} test cases, reference fields u_i, and predictions $\widehat{u}_i$, the reported mean relative L_2 error is

$$\frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} \frac{\|\widehat{u}_i - u_i\|_2}{\|u_i\|_2}, \tag{14}$$

where i indexes cases and $\|\cdot\|_2$ is the Euclidean norm of sampled field values. Evaluation retains native channel/time aggregation and reverses output normalization where applicable. AirfRANS uses

its native geometric inputs and ensemble evaluation; its volume and surface errors measure pressure in physical units, not normalized multi-channel MSE. ShapeNet-Car uses single-graph inputs and predicts velocity and pressure. Table 2 reports the field and coefficient metrics; lift and drag follow native aerodynamic postprocessing, and Spearman correlations are computed across test geometries.

C.3 Execution Schedules and Parameter Counts

All eight-layer looped models use $1 + 3 \times 2 + 1$: one independent prefix layer, three core positions executed twice, and one independent suffix layer. This includes the main results, ablations, eight-layer depth-study models, and the separate three-expert Airfoil parameter comparison. The 16-, 24-, 32-, and 40-layer models retain two prefix layers, two suffix layers, and two core repetitions, with 6, 10, 14, and 18 stored core positions, respectively.

A width-F expert contains $2CF + F + C$ parameters, including biases; the affine router contributes $E(C + 1)$. For J stored positions and R repetitions, the core parameter count is

$$P_{\text{core}} = J\big[P_{\text{back}} + E(2CF + F + C)\big] + RJP_{\text{visit}}, \tag{15}$$

where P_{back} counts a position's shared input, value, and output transformations, and P_{visit} counts one execution's independent slicing/deslicing projections, temperatures, router, and normalizations. Complete-model counts additionally include lifting, prefix/suffix layers, final normalization, and readout.

From 8 to 40 layers, the number of independently parameterized backbone/expert blocks increases from $1 + 3 + 1 = 5$ to $2 + 18 + 2 = 22$, while the number of visit-specific interfaces increases by 32. The parameter increment is

$$\Delta P_{\text{ours}} = 17\big[P_{\text{back}} + E(2CF + F + C)\big] + 32P_{\text{visit}}. \tag{16}$$

An unshared baseline adds 32 complete layers. We report $100(1 - \Delta P_{\text{ours}}/\Delta P_{\text{baseline}})\%$, using the task settings in Table 5. Relative to Transolver and LinearNO, respectively, the added parameters decrease by 66.48% and 46.26% on Darcy, and by 51.21% and 21.76% on Airfoil. On Elasticity, four experts instead increase the increment by 7.08% and 32.04%. These architecture-based comparisons depend on the specified schedule and sharing pattern.

C.4 Depth Scaling

Table 7 gives all measurements plotted in Figure 4. Each depth is trained separately, with fixed channel and latent widths and the expert counts in Table 5.

Table 7: Test relative L_2 errors in the depth study. Columns give the number of layers. The Transolver Elasticity result at 8 layers is a separate depth-study record from Table 1.

Dataset	Model	8	16	24	32	40
Airfoil	Transolver	0.0053	0.0055	0.0046	0.0043	0.0036
	LinearNO	0.0048	0.0042	0.0039	0.0037	0.0034
	PhyLoopNO	0.0046	0.0043	0.0039	0.0036	0.0032
Darcy	Transolver	0.0055	0.0053	0.0051	0.0048	0.0047
	LinearNO	0.0050	0.0046	0.0040	0.0038	0.0034
	PhyLoopNO	0.0048	0.0044	0.0042	0.0039	0.0034
Elasticity	Transolver	0.0064	0.0056	0.0049	0.0047	0.0047
	LinearNO	0.0054	0.0048	0.0046	0.0045	0.0043
	PhyLoopNO	0.0047	0.0049	0.0044	0.0042	0.0041

C.5 Parameter-Sharing Ablations

Table 8 supplies the data for Figure 2. Both looped variants use the $1 + 3 \times 2 + 1$ eight-layer schedule and one FFN per layer. Full-share ties all core parameters across repetitions. Selective-share keeps slicing/deslicing projections, their temperatures, and both LayerNorms independent, while sharing the remaining parameters. The radar coordinate is $1 -$ model error$/$LinearNO error, with the baseline

from Table 1. Table 3 isolates slicing/deslicing sharing in the full expert model: it ties each projection and its temperature across repetitions, while routers and LayerNorms remain independent. Q and K remain distinct maps. Both rows use the expert counts selected for Table 1.

Table 8: Test relative L_2 errors underlying Figure 2. Both looped variants use a single FFN.

Configuration	Airfoil	Pipe	Plasticity	Darcy	NS	Elasticity
LinearNO	0.0048	0.0028	0.0013	0.0050	0.0752	0.0054
Looped: full sharing	0.0060	0.0046	0.0021	0.0055	0.1170	0.0082
Looped: independent slicing/deslicing	0.0050	0.0041	0.0015	0.0048	0.0991	0.0070

D VISUALIZATIONS

Weight maps. Figures 5–8 visualize the slicing (key) weights K and deslicing (query) weights Q of PHYLOOPNO on Pipe, Airfoil, Darcy, and Elasticity. As in Equation (1), K determines how spatial points contribute to each latent summary, while Q determines how that summary contributes to the reconstructed point features. Each mosaic shows the 64 latent states of the first attention head. Each state map is divided by its own spatial maximum for display; colors therefore show relative spatial patterns, not absolute weight magnitudes across states or between K and Q.

The maps contain both localized and extended responses. Pipe shows concentrated key weights and broader query patterns along the channel; Airfoil exhibits patterns around the body and downstream. Darcy combines boundary-focused and interior responses, while Elasticity shows variations around the opening and outer boundary. The different patterns of K and Q illustrate their distinct roles in aggregation and reconstruction. In PHYLOOPNO, the projections that produce these weights remain independent across loop iterations, allowing spatial interactions to adapt while the core transformations are reused.

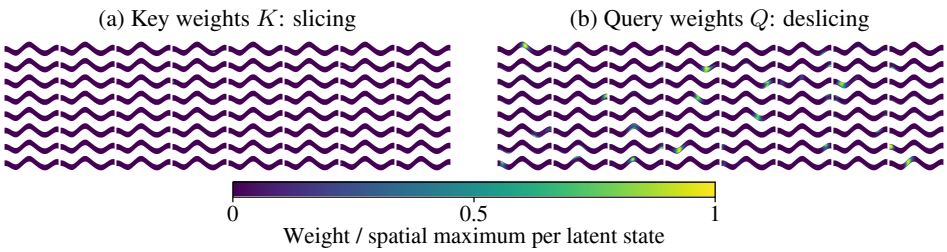

Figure 5: Slicing and deslicing weight maps of PHYLOOPNO on Pipe.

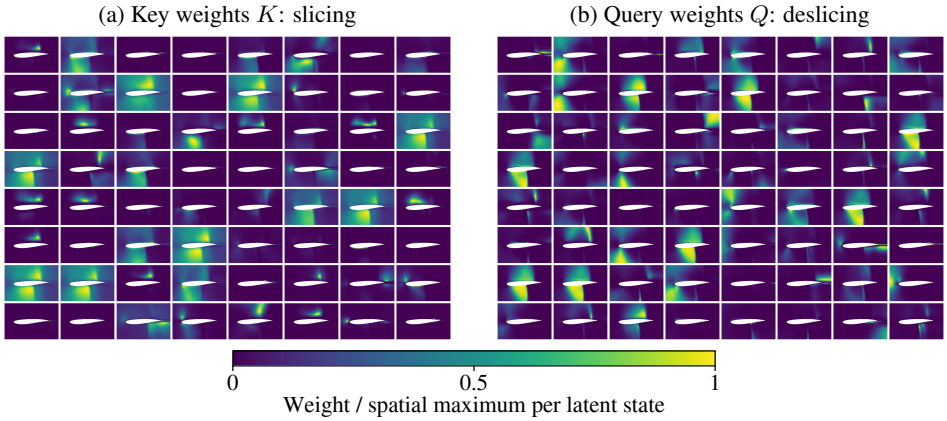

Figure 6: Slicing and deslicing weight maps of PHYLOOPNO on Airfoil.

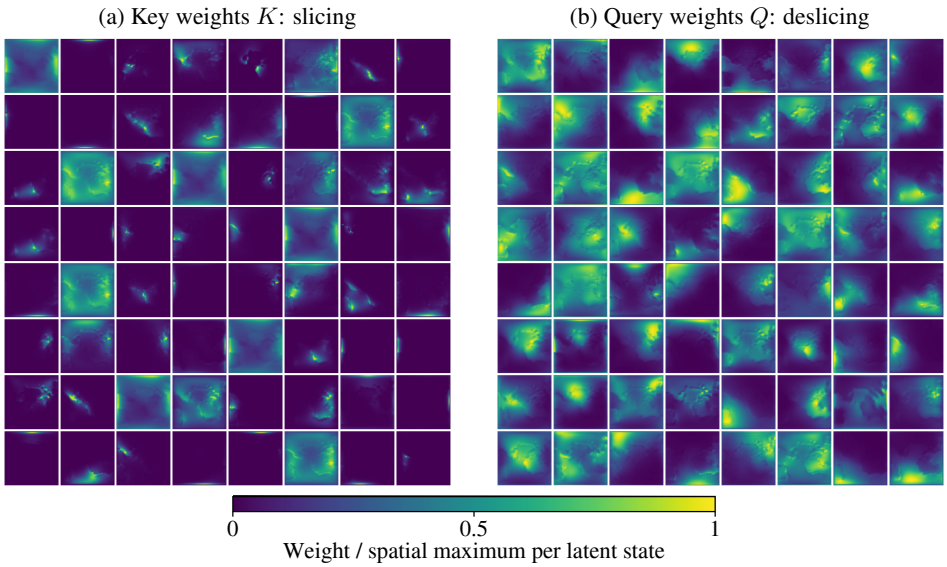

Figure 7: Slicing and deslicing weight maps of PHYLOOPNO on Darcy.

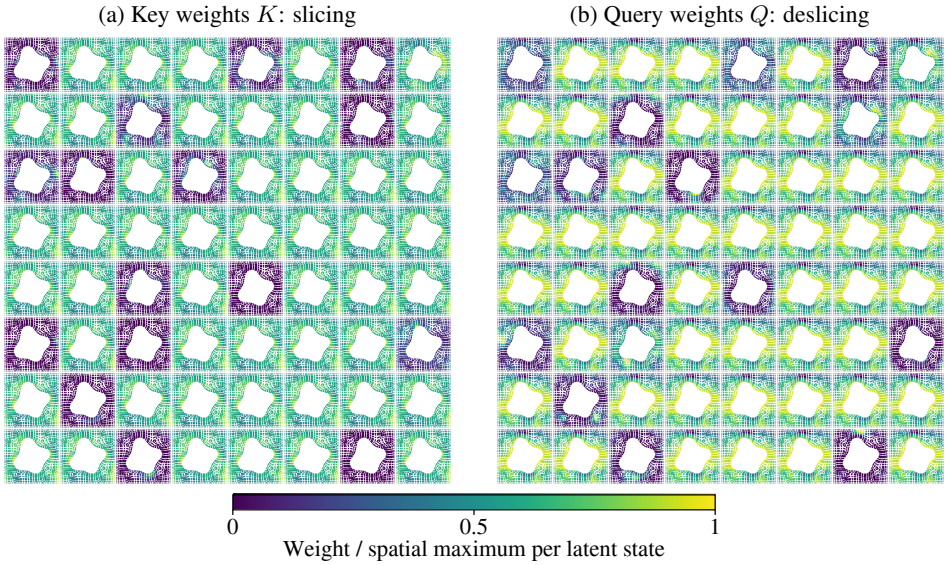

Figure 8: Slicing and deslicing weight maps of PHYLOOPNO on Elasticity.

D.1 Expert Gate Maps

Figure 9 visualizes the physics-gating mechanism in Eq. (4) at the three stored core positions of the $1 + 3 \times 2 + 1$ model. Core 00, 01, and 02 correspond to $j = 1, 2, 3$. At every spatial point, $\sum_{e=1}^{E} \pi_e = 1$; uniform mixing gives $1/E$, namely $1/4$ for Airfoil and Elasticity and $1/3$ for Pipe. Brighter colors indicate larger mixing weights on the common 0–1 scale. All experts evaluate every point, and the gate weights determine how their outputs are combined.

At a fixed core position, the two visits reuse the same expert parameters. The input features and router differ between visits, allowing their weight maps to change. Expert labels are local to each core position; different positions have distinct expert banks. The maps illustrate spatially varying mixtures and their adaptation across visits.

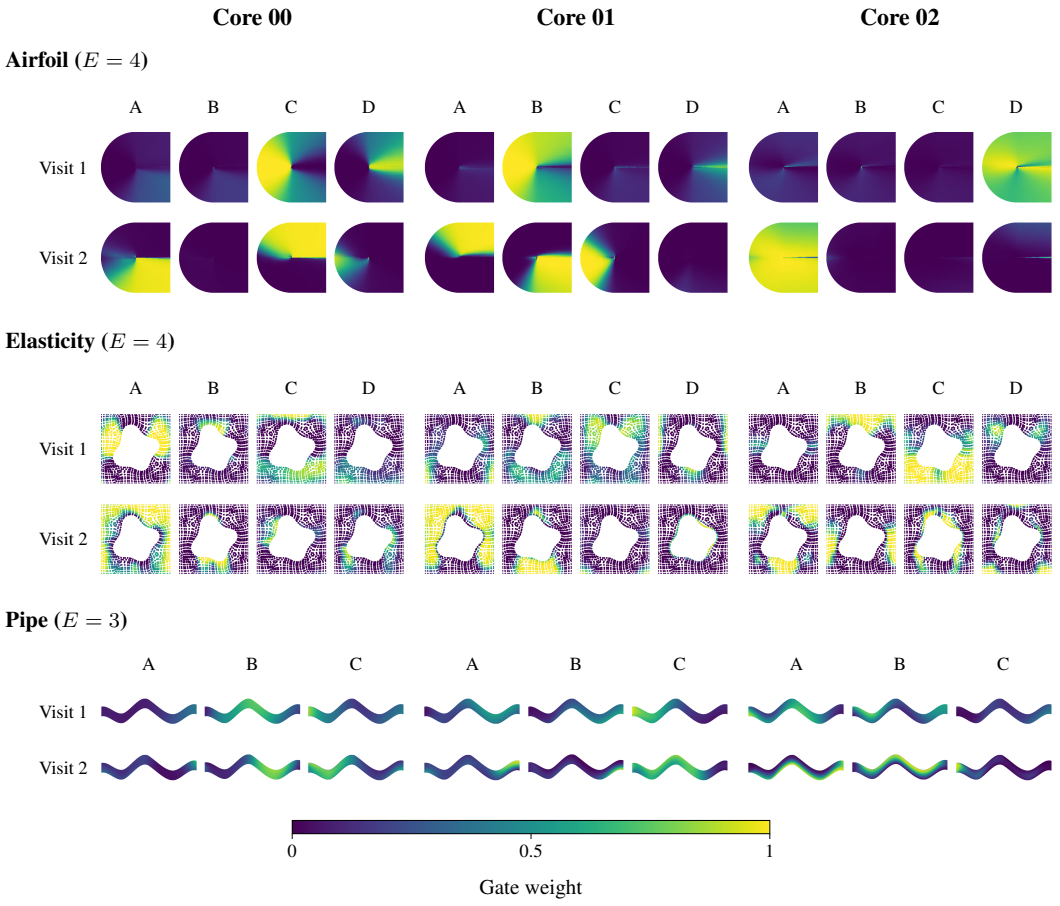

Figure 9: Spatial expert gate weights on Airfoil, Elasticity, and Pipe. Each dataset occupies one row of core panels, ordered 00, 01, and 02 from left to right. Within each panel, the upper and lower maps show Visits 1 and 2; columns index experts A–D for Airfoil and Elasticity and A–C for Pipe. All panels share the fixed 0–1 color scale.