ONE-WAY COUPLING VIA PATCHWISE ADAPTIVE NORMALIZATION IN TRANSFORMER SURROGATES

Anonymous authors
Paper under double-blind review

ABSTRACT

Many physical systems exhibit *one-way coupling*: exogenous fields such as medium density or obstacle geometry influence the dynamics without being affected by them. We investigate how explicitly representing these fields can improve the accuracy and computational efficiency of transformer-based PDE surrogates. Building on spatially adaptive normalization, we introduce **Patchwise Adaptive Normalization** (`PAdaNorm`), which partitions exogenous fields into patches aligned with the state tokenization. We evaluate `PAdaNorm` on *datasets from three different domains*, using a Walrus-inspired transformer architecture. Across the three PDE families, `PAdaNorm` is on par with or up to 16% better than parameter-matched baselines in mean long-horizon variance-scaled RMSE on 60-step autoregressive rollouts. We also find that `PAdaNorm` better maintains hard physical constraints over long rollouts, such as preventing flow into impenetrable solid regions in porous media. When the conditioning is static, `PAdaNorm` caches the conditioning embeddings once and reuses them across the entire rollout, reducing inference FLOPs per step by up to 36% compared with parameter-matched baselines. An ablation over eight conditioning variants shows that modulating a single attention axis (temporal or spatial) suffices and that a lightweight shared MLP across blocks remains competitive with per-block projections.

1 INTRODUCTION

Transformer-based surrogates have shown promise for learning the evolution of physical fields across diverse systems governed by partial differential equations (PDEs) (McCabe et al., 2024; Herde et al., 2024; McCabe et al., 2026). Maintaining prediction accuracy over long autoregressive rollouts remains challenging, as errors introduced at individual steps can propagate and accumulate (Brandstetter et al., 2022; Lippe et al., 2023; Koehler et al., 2024). One contributing factor is that the model is trained on ground-truth input frames but, during a rollout, receives its own earlier predictions as inputs, whose distribution slowly shifts away from the one seen in training (Bengio et al., 2015; Sanchez-Gonzalez et al., 2020). Existing approaches mitigate this rollout degradation through input noise during training (Sanchez-Gonzalez et al., 2020; Pfaff et al., 2021), training over multiple prediction steps (Brandstetter et al., 2022; Koehler et al., 2024), or patch jittering (McCabe et al., 2026). Alongside these approaches, we investigate how conditioning on known spatial inputs, such as material properties and obstacle geometry, affects autoregressive prediction accuracy.

In many PDE models, the evolving state depends on spatial inputs such as material properties, obstacle geometry, or externally imposed fields. We refer to these inputs as *exogenous fields* when they influence the predicted dynamics without being affected by them, giving rise to *one-way coupling*. For example: (i) thermal stress in a material where a temperature distribution causes the material to expand and develop internal mechanical stresses, while mechanical stress does not impact the temperature profile; or (ii) quantum physics, whereby an external potential influences the dynamics of a particle; or (iii) in photonics, where the refractive index of the material bends the path of light rays.

Existing PDE surrogates incorporate information about the physical system through several complementary mechanisms. Models such as $\text{GP}_{\text{hy}}\text{T}$ use histories of physical states as context from which to infer the underlying dynamics (Wiesner et al., 2026). When conditioning fields are explicitly available, they can be concatenated with the state channels, preserving their spatial correspondence at the input. Other architectures introduce separate conditioning pathways: GNOT uses heterogeneous

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.

cross-attention to incorporate multiple input functions (Hao et al., 2023), while GINOT encodes geometry and integrates it with query-point representations through cross-attention (Liu et al., 2026). Adaptive normalization provides another mechanism for injecting conditioning into intermediate features: instead of learning fixed scale and shift parameters, a normalization layer predicts them from a conditioning signal, as in the adaptive layer normalization (AdaLN) of the Diffusion Transformer (Peebles & Xie, 2023). Poseidon uses lead-time-conditioned layer normalization (Herde et al., 2024), whereas UPT employs DiT-style scale, shift, and gate modulation to condition on time and boundary-condition parameters such as inflow velocity (Alkin et al., 2024; Peebles & Xie, 2023).

Spatially varying modulation has precedents in both image synthesis and physical modeling. SPADE generates spatial scale and shift parameters from semantic segmentation masks to modulate convolutional feature maps (Park et al., 2019). FlowAR's Spatial-AdaLN produces position-specific scale, shift, and gate parameters from aligned semantic representations within a transformer (Ren et al., 2025). P3D uses region tokens derived from globally aggregated state features to modulate local decoder regions through adaptive normalization (Holzschuh et al., 2026). These methods establish spatial modulation as a mechanism for structured conditioning. Building on this principle, we investigate a specific design for autoregressive PDE surrogates: independently encoding exogenous field patches aligned with the state tokenization, sharing the nonlinear encoder across blocks, and using block-specific projections to modulate the spatial attention and feed-forward block using spatially-aware conditioning.

We call this design **Patchwise Adaptive Normalization** (`PAdaNorm`; Fig. 1), a conditioning approach for one-way coupled fields in physical PDEs. `PAdaNorm` conditions the transformer on spatially-varying material properties (e.g., density, refractive index, potential strength, ...) at the granularity of individual patches. The conditioning fields are partitioned into the same non-overlapping patches as the input state, and each patch is independently mapped through a shared MLP to produce a conditioning vector. This yields one conditioning vector per spatial patch, capturing the local effect of the coupled exogenous variables within each patch's receptive field. These patchwise conditioning vectors then drive an adaLN-Zero-style shift, scale, and gate modulation (Peebles & Xie, 2023) of the transformer's intermediate representations on a per-patch basis: each patch token receives its own shift, scale, and gate parameters derived from the conditioning fields at that spatial location. For static conditioning fields and fixed patch alignment, all modulation parameters can be computed once and reused throughout, amortizing parameter-generation cost over the rollout.

We evaluate `PAdaNorm` on several PDE modeling tasks, spanning different PDE families (acoustic scattering, 2D Navier-Stokes, and CO_2–water flow in porous media). Our experiments demonstrate that `PAdaNorm` outperforms or closely matches existing baselines, with a comparable parameter count and with fewer forward-pass FLOPs than parameter-matched baselines (App. B.5). We also find that `PAdaNorm` better maintains hard physical constraints over long rollouts: on CO_2–water flow in porous media, it lets significantly less fluid into the impenetrable rock than every baseline (Sec. 4). Crucially, when the conditioning variables are static, each rollout step of `PAdaNorm` costs the same matrix-multiplication FLOPs as the same transformer without conditioning; only the element-wise modulation remains.

2 Related Works

Neural surrogates for PDEs Neural operators learn mappings between physical fields, with the Fourier Neural Operator parameterizing integral kernels in Fourier space to model families of PDE solutions (Li et al., 2021). Recent work has expanded toward pretraining across heterogeneous physical systems. MPP learns transferable representations through a shared transformer backbone (McCabe et al., 2024), Poseidon uses a multiscale operator transformer (Herde et al., 2024), and DPOT introduces autoregressive denoising pretraining (Hao et al., 2024). Walrus further develops this direction by introducing a transformer that supports diverse physical fields in two and three spatial dimensions (McCabe et al., 2026). Our work investigates conditioning mechanisms within a Walrus-inspired backbone, focusing on how spatially varying exogenous fields can condition its intermediate representations.

Conditioning mechanisms Feature-wise Linear Modulation (Perez et al., 2018) formalizes the general idea of conditioning neural networks by predicting per-feature scale and shift parameters from

an auxiliary signal, connecting a family of prior adaptive normalization methods (Dumoulin et al., 2018). In diffusion models, Dhariwal & Nichol (2021) use Adaptive Group Normalization (AdaGN) to condition U-Net layers on timestep and class embeddings. The Diffusion Transformer (Peebles & Xie, 2023) adapts this mechanism for transformers with AdaLN-Zero, modulating the LayerNorm parameters based on a conditioning vector and initializing each residual block as the identity through zero-initialized gating. Crucially, standard AdaLN applies uniform modulation to all tokens (Chen et al., 2024), which suffices for global signals such as timesteps or class labels but creates a representational bottleneck when the conditioning varies spatially. In image generation, SPADE (Park et al., 2019) predicts spatially varying scale and shift from semantic segmentation masks to modulate convolutional features, and FlowAR (Ren et al., 2025) and PixelDiT (Yu et al., 2026) bring such spatially adaptive modulation to transformers. `PAdaNorm` builds on these mechanisms by independently encoding exogenous field patches aligned with the state tokenization. Its nonlinear patch encoder is shared across transformer blocks, while block-specific linear projections generate the modulation parameters. For static fields and fixed patch alignment, these parameters can be cached throughout an autoregressive rollout.

Conditioning mechanisms for PDE surrogates The simplest approach, used by Walrus (McCabe et al., 2026) and MPP (McCabe et al., 2024), treats conditioning variables as additional input channels concatenated to the state tokens (*in-context conditioning*), without any architectural inductive bias to disentangle state from forcing variables. Cross-attention is an alternative, allowing the system's state tokens to attend over the conditioning fields, at a quadratic cost in the number of patches (App. B.5) and without enforcing spatial or temporal locality. Poseidon (Herde et al., 2024) conditions on continuous time through time-dependent LayerNorm (Ba et al., 2016) parameters, enabling evaluation at arbitrary temporal resolution. PDE-Transformer (Holzschuh et al., 2025) uses AdaLN-Zero to modulate transformer blocks based on a learned PDE-type embedding, while UPT conditions on time and boundary-condition parameters (Alkin et al., 2024). In recent work, Holzschuh et al. (2026) propose P3D, which applies adaptive instance normalization to local decoder regions using region tokens derived from globally aggregated state features. `PAdaNorm` instead derives each patch's modulation directly from the corresponding exogenous field patch, independently of the evolving state. This imposes locality on the conditioning pathway while allowing the transformer's attention to model interactions between spatial locations.

3 Method

3.1 Walrus Transformer

Building on its success as a foundation physics model, we rely on a minimal version of Walrus (McCabe et al., 2026), retaining its core components to minimize confounding factors. We depart from the original implementation in three ways. First, we include neither patch jitter nor adaptive-compute patching. Second, temporal position is encoded with a 1D rotary embedding rather than the learned T5-style relative position encoding. Third, temporal attention carries no causal mask, which was originally found to have no impact on performance compared to causal masking (McCabe et al., 2026, App. B.3).

Our implementation of Walrus differs from the vanilla transformer as it includes successive (i) temporal attention and (ii) spatial attention blocks at each layer. See App. A.1 for details on spatial and temporal attention and McCabe et al. (2026) for a full description of the Walrus architecture. Formally, the model predicts the evolution of a 2D state $\mathbf{u}(x, y; \mathbf{C}; t + \Delta t)$ at a fixed timestep Δt and subject to the spatially varying, static one-way coupled fields $\mathbf{C}(x, y)$ (of dimension C_{cond}) given T_{in} consecutive snapshots of the system state, separated by fixed intervals Δt. We restrict our experiments to static fields, but the formulation extends directly to time-varying fields, $\mathbf{C}(x, y, t)$, by computing a set of modulation parameters at each input timestep (Sec. 6). We consider both $\mathbf{u}$ and $\mathbf{C}$ fields to be discretized on an $H \times W$ spatial grid. Each input snapshot is partitioned into a collection of $N_x \times N_y$ square patches of size $P \times P$, with $N_x = H/P$ and $N_y = W/P$, which are each embedded into the model's latent space of embedding dimension D.

3.2 ADAPTIVE LAYERNORM

Rather than learning static gain and bias parameters as in standard Layer Normalization (Ba et al., 2016), `AdaLN`, as introduced in the Diffusion Transformer (Peebles & Xie, 2023), predicts the scale and shift from an external conditioning embedding $\mathbf{c} \in \mathbb{R}^{D_c}$.

Formally, let $\mathbf{h} \in \mathbb{R}^D$ denote the embedding of a single patch at a single timestep. We project the conditioning vector $\mathbf{c}$ via a linear layer to obtain token-independent scale, shift, and gate vectors, denoted as $\boldsymbol{\gamma}(\mathbf{c}), \boldsymbol{\beta}(\mathbf{c}), \boldsymbol{\alpha}(\mathbf{c}) \in \mathbb{R}^D$, respectively:

$$\boldsymbol{\gamma}(\mathbf{c}) = \mathbf{W}_\gamma \mathbf{c} + \mathbf{b}_\gamma, \quad \boldsymbol{\beta}(\mathbf{c}) = \mathbf{W}_\beta \mathbf{c} + \mathbf{b}_\beta, \quad \boldsymbol{\alpha}(\mathbf{c}) = \mathbf{W}_\alpha \mathbf{c} + \mathbf{b}_\alpha, \tag{1}$$

where $\mathbf{W}_\gamma, \mathbf{W}_\beta, \mathbf{W}_\alpha \in \mathbb{R}^{D \times D_c}$ and $\mathbf{b}_\gamma, \mathbf{b}_\beta, \mathbf{b}_\alpha \in \mathbb{R}^D$ are learnable projection weights and biases. The scale and shift then give the adaptive normalized representation

$$\texttt{AdaLN}(\mathbf{h}, \mathbf{c}) = (\mathbf{1} + \boldsymbol{\gamma}(\mathbf{c})) \odot \texttt{LayerNorm}(\mathbf{h}) + \boldsymbol{\beta}(\mathbf{c}), \tag{2}$$

where $\mathbf{1}$ represents a vector of ones, while the gate $\boldsymbol{\alpha}(\mathbf{c})$ scales the sublayer output before it re-enters the residual stream, as made explicit in Sec. 3.3.

3.3 PATCHWISE ADAPTIVE NORMALIZATION

The above-described standard implementation of AdaLN applies the same modulation uniformly to all spatial tokens via a single global conditioning vector $\mathbf{c}$. This is appropriate when the conditioning signal is spatially homogeneous (e.g., a scalar timestep or a global parameter), but limiting when the conditioning fields exhibit spatial structure, e.g., heterogeneous material properties (density, refractive index, or heat capacity).

We introduce *Patchwise Adaptive Normalization* (`PAdaNorm`), which produces spatially varying modulation parameters by operating at the granularity of spatial patches. Given C_{cond} spatially varying conditioning fields stacked as $\mathbf{C}(x, y) \in \mathbb{R}^{C_{\text{cond}} \times H \times W}$, we partition them into the same $N_x \times N_y$ grid of non-overlapping patches of size $P \times P$ used for tokenizing the input state. Let $s = (i, j) \in [1, N_x] \times [1, N_y]$ index a patch, $\mathcal{P}_s$ denote its $P \times P$ pixel region, and $\mathbf{C}|_{\mathcal{P}_s} \in \mathbb{R}^{C_{\text{cond}} \times P \times P}$ the sub-tensor of the conditioning fields restricted to that region. Each patch is flattened into a vector of dimension $C_{\text{cond}} P^2$ and mapped through a shared encoder f_θ to produce a local conditioning vector:

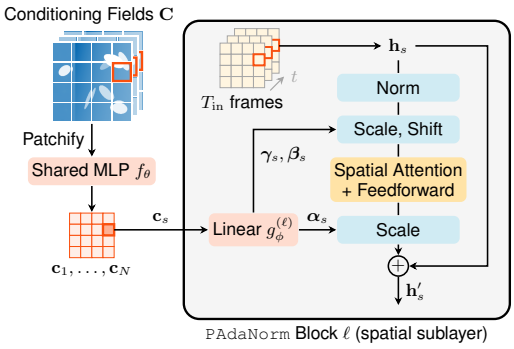


Figure 1: **Patchwise Adaptive Normalization**. The spatially varying conditioning fields $\mathbf{C}$ are partitioned into patches aligned with the input tokenization. Each patch is independently mapped through a shared MLP f_θ to produce a local conditioning vector $\mathbf{c}_s$. Each token $\mathbf{h}_s$, in every frame of the history window, is then modulated by the scale, shift, and gate derived from $\mathbf{c}_s$ (conditioning vector of the corresponding patch).

$$\mathbf{c}_s = f_\theta(\text{vec}(\mathbf{C}|_{\mathcal{P}_s})) \in \mathbb{R}^{D_c}, \tag{3}$$

where $\text{vec}(\cdot)$ denotes the flattening operator. In practice, f_θ is an MLP with hidden width D_h, output dimension D_c, and SiLU activations. f_θ is shared across all patches and all blocks, and evaluated once per forward pass. This yields a set of patch-local conditioning vectors $\{\mathbf{c}_s\}_{s \in [1,N_x] \times [1,N_y]}$, each depending only on the conditioning values within its corresponding patch.

Each block ℓ then maps these conditioning vectors to per-patch scale, shift, and gate parameters through its own projection $g_\phi^{(\ell)}$, a single linear layer:

$$\boldsymbol{\gamma}_s^{(\ell)}, \ \boldsymbol{\beta}_s^{(\ell)}, \ \boldsymbol{\alpha}_s^{(\ell)} = g_\phi^{(\ell)}(\mathbf{c}_s) \in \mathbb{R}^D \times \mathbb{R}^D \times \mathbb{R}^D. \tag{4}$$

These modulate the single normalization and the single residual gate of an attention or feed-forward block; our main implementation modulates only the spatial attention block (*spatial-only* configuration, Fig. 1), although the modulation can be placed on any block that is preceded by a normalization and

followed by a residual gate (App. B.3). Writing $\mathbf{h}_s$ for the token embedding at spatial position s entering block ℓ, and $\tilde{\mathbf{H}}$ for the collection of modulated tokens at a given history timestep, we define:

$$\begin{aligned} \tilde{\mathbf{h}}_s &= \left(\mathbf{1} + \boldsymbol{\gamma}_s^{(\ell)}\right) \odot \texttt{Norm}(\mathbf{h}_s) + \boldsymbol{\beta}_s^{(\ell)}, \\ \mathbf{h}'_s &= \mathbf{h}_s + \boldsymbol{\alpha}_s^{(\ell)} \odot \left[\texttt{Attn}(\tilde{\mathbf{H}})_s + \texttt{FFN}(\tilde{\mathbf{h}}_s)\right], \end{aligned} \tag{5}$$

where $\texttt{Attn}(\cdot)$ and $\texttt{FFN}(\cdot)$ are the two parallel branches of the spatial attention block (Eq. 9), $\texttt{Norm}$ is a normalization layer (e.g. $\texttt{LayerNorm}$ or $\texttt{RMSGroupNorm}$), and $\mathbf{1}$ is a vector of ones.

Following the AdaLN-Zero strategy, each $g_\phi^{(\ell)}$ is initialized so that every block starts close to the identity and early training remains stable. Also, because the conditioning fields we consider are static, they are identical at every timestep of the input window, so a single set of modulation parameters $(\boldsymbol{\gamma}_s^{(\ell)}, \boldsymbol{\beta}_s^{(\ell)}, \boldsymbol{\alpha}_s^{(\ell)})$ per block is shared by all T_{in} timesteps.

Crucially, f_θ is shared across all spatial patches *and* all blocks, while each block owns a separate projection $g_\phi^{(\ell)}$; Appendix B.3 reports the variants that share g_ϕ across blocks as well. The number of conditioning parameters is therefore *independent* of the number of spatial patches and grows only linearly with depth; the conditioning compute, unlike cross-attention, grows linearly with the number of patches and does not depend on the history length T_{in}. A single evaluation of f_θ and $g_\phi^{(\ell)}$ can therefore be cached and reused through the entire rollout; App. B.5 derives the cost of each conditioning strategy and reports measured FLOP counts.

In our default implementation, conditioning patches are non-overlapping and aligned one-to-one with state patches, so each token $\mathbf{h}_s$ receives the conditioning vector $\mathbf{c}_s$ of its own patch. $\texttt{PAdaNorm}$ only requires, however, that every token be assigned one conditioning vector: a vector can be shared by several neighboring tokens (coarse patches) or computed from a window that overlaps neighboring patches, both of which we study in Sec. 5.2. At the other extreme, if a single conditioning vector $\mathbf{c}$ is shared by all tokens, $\texttt{PAdaNorm}$ reduces to the global AdaNorm baseline of App. C.1.

4 Experiments

4.1 Experimental Settings

Datasets We evaluate on five public datasets from three sources. From The Well (Ohana et al., 2024), we use three acoustic scattering datasets; from FlowBench (Tali et al., 2025), the Flow Past Objects dataset; and CO_2–water flow in porous media from Abdellatif et al. (2026). *i) Acoustic scattering (The Well):* the Maze, Inclusions, and Discontinuous datasets simulate acoustic wave propagation through heterogeneous media at 256×256 resolution. State variables are pressure and velocity (3 channels), and one-way coupled conditioning fields are density and speed of sound (2 channels), whose spatial structure defines the scattering geometry. In the *Joint* setting, a single model is trained on all three acoustic datasets simultaneously. *ii) Flow Past Objects (FlowBench):* 2D Navier–Stokes simulations at 1024×256 resolution, where state fields are the two velocity components and pressure (3 channels), and conditioning channels are the obstacle binary mask, signed distance function, and Reynolds number. *iii) CO_2–water flow in porous media:* simulates supercritical CO_2 injected into a water-saturated porous rock at the pore scale. State fields are the water saturation, pressure, and the two x and y velocity components; conditioning channels are the binary pore mask separating pore space from rock grains and two normalized coordinate channels. This setting is complementary to the two above in that a) it is a different PDE family, and b) its physics are considerably more complex, as the CO_2 front advances along the path of least resistance through the rock, so information across the entire front must be aggregated to determine where it progresses next.

Benchmark We compare $\texttt{PAdaNorm}$ against several conditioning strategies. The *In-context* baselines concatenate the conditioning fields to the input state as additional channels before patch embedding, as in Walrus (McCabe et al., 2026). To control for the parameter overhead introduced by the $\texttt{PAdaNorm}$ conditioning MLPs, we include parameter-matched In-context variants that increase either model width or depth to match the $\texttt{PAdaNorm}$ parameter count. We also compare against a global *AdaNorm* baseline (see App. C.1), which average-pools the conditioning fields and maps

the result, through a single MLP shared across all blocks, to spatially uniform scale, shift, and gate parameters that modulate both attention sublayers. We further compare against a (more compute-intensive) *cross-attention* baseline, in which a dedicated encoder tokenizes the conditioning fields on the same patch grid, and all state patches (in space and time) attend to the conditioning patches using spatial 2D RoPE. Finally, we include GNOT (Hao et al., 2023), a neural operator transformer baseline that uses linear attention with heterogeneous cross-attention for conditioning; we use a patch tokenization, *GNOT*, which reuses our patch embedding for a fair compute comparison. All models are evaluated via 60-step autoregressive rollouts. We report the variance-scaled RMSE (vRMSE) (Ohana et al., 2024), the RMSE normalized by the spatial standard deviation of the ground truth,

$$\mathrm{vRMSE}(\mathbf{u}, \hat{\mathbf{u}}) = \left(\frac{\langle |\mathbf{u} - \hat{\mathbf{u}}|^2 \rangle}{\langle |\mathbf{u} - \bar{\mathbf{u}}|^2 \rangle + \epsilon} \right)^{1/2}, \tag{6}$$

where $\mathbf{u}$ is the ground truth, $\hat{\mathbf{u}}$ the prediction, $\langle \cdot \rangle$ the spatial mean, $\bar{\mathbf{u}} = \langle \mathbf{u} \rangle$, and $\epsilon = 10^{-7}$. We evaluate Eq. 6 and average over channels and validation windows to obtain one value per rollout step, and then aggregate into three horizons: first step $T{=}1$, short $T{\in}[1{:}20]$, and long horizon $T{\in}[21{:}60]$. For each run, we select the checkpoint with the lowest $T{\in}[21{:}60]$ validation vRMSE and report all metrics on the same validation windows, with the compute budget fixed in advance at 50,000 steps.

Hyperparameter settings All models use the Walrus-inspired architecture described in Sec. 3.1 and are trained for 50,000 steps with AdamW and a per-field L_1 loss, except on CO_2, which uses an L_2 loss. On the acoustic datasets, models use a patch size of 8, an embedding dimension of 256, and 10 transformer blocks; on FlowBench, a patch size of 16 with an embedding dimension of 384; on CO_2, a patch size of 8 with an embedding dimension of 384 and 10 blocks, with four input timesteps. The `PAdaNorm` conditioning encoder is a 4-layer MLP with hidden dimension 512, shared across patches and blocks. Full configuration details, including parameter-matched model sizes, are provided in App. B.4.

4.2 Results

Comparison with in-context conditioning At comparable parameter counts, we obtain the clearest accuracy gains on the acoustic datasets by allocating capacity to `PAdaNorm`'s spatial conditioning. We compare the default spatial-only configuration with wider and deeper In-context models that receive the conditioning fields as additional input channels (Table 1). Across three seeds, we reduce mean long-horizon vRMSE by 8–16% relative to the strongest parameter-matched In-context baseline on each acoustic dataset and in the Joint setting, where a single model is trained on all three. The improvement therefore comes from the dedicated conditioning pathway rather than from the additional parameters it introduces. On FlowBench, we obtain lower mean errors than the In-context baselines at all three reported horizons, although the long-horizon differences are small relative to the seed variation (Table 5). On CO_2, we find no clear accuracy advantage over input concatenation (Table 6). Representative rollouts on each domain are shown in App. B.2.

Comparison with alternative conditioning mechanisms We next compare `PAdaNorm` with global adaptive normalization and cross-attention (Table 1). We obtain lower mean long-horizon errors than global AdaNorm on the acoustic and FlowBench datasets, with the largest differences on acoustic scattering. In contrast, on CO_2, global AdaNorm attains the lowest mean long-horizon error. This supports the effectiveness of the patchwise configuration, although the two approaches also differ in modulation placement and parameter count. Compared with cross-attention, we obtain lower mean errors on all five datasets and in the Joint setting. Allowing state tokens to attend over the conditioning fields therefore does not provide a consistent accuracy advantage over `PAdaNorm`, despite requiring more computation. We also obtain lower long-horizon errors than GNOT, which uses a different architecture. We examine computational costs below and compare spatial-only conditioning with other `PAdaNorm` variants in the ablations.

Hard constraints on CO_2 Although long-horizon errors on CO_2 are close across models (Table 6), the dataset reveals a difference that vRMSE alone does not capture. It allows us to test how well each conditioning mechanism enforces a *hard constraint*: CO_2 and water should flow *around* the impermeable rock grains, never into them. Measuring the mean absolute error on rock pixels alone

(Tab. 7) shows that PAdaNorm is best at enforcing this constraint, with the lowest error on all four fields on average across the three seeds. Interestingly, global AdaNorm ranks second on three of the four fields, although its modulation carries no spatial information, suggesting that part of the advantage stems from injecting the conditioning at every block rather than only at the input.

Computational efficiency Across all datasets, we achieve improved or competitive accuracy with consistently lower rollout FLOPs than parameter-matched In-context baselines and cross-attention models (Table 12, App. B.5). Because the conditioning fields we consider are static, PAdaNorm's conditioning embeddings and modulation parameters can be computed once and reused throughout inference. Cross-attention can also reuse the conditioning embeddings and their key and value projections, but must recompute the state-dependent queries, attention products, and output projections. Accounting for reusable computations in both methods, we use 29–36% fewer FLOPs than cross-attention over a 60-step rollout. The computational benefit of PAdaNorm is not limited to caching: even when recomputing the conditioning parameters, we require 30–35% fewer forward-pass FLOPs than the evaluated cross-attention configurations. With caching, PAdaNorm's total rollout cost remains approximately equal to that of the original, smaller In-context model. We therefore improve accuracy on acoustics and FlowBench at little additional rollout cost, while on CO_2 we achieve comparable accuracy with less computation than the wider and deeper controls and cross-attention.

Table 1: Long-horizon validation vRMSE (↓, $T \in [21{:}60]$) for baselines and PAdaNorm spatial-only, reported as mean ± standard deviation over three seeds. Best mean in **bold** and second best underlined within each dataset. ‡Parameter-matched In-context controls. The *Joint* result reports vRMSE averaged over the three datasets, for a single model trained on all three simultaneously. GFLOPs denotes one forward pass at batch size one; cached rollout costs are reported in Table 12.

	In-context (base)	In-context (wide)‡	In-context (deep)‡	AdaNorm (global)	Cross-attention	GNOT	PAdaNorm (ours)
Acoustic scattering (The Well (Ohana et al., 2024))							
#Params	13.3 M	18.0 M	18.6 M	15.5 M	18.5 M	18.8 M	19.1 M
GFLOPs	113.5	138.9	158.6	113.3	179.7	41.2	125.2
Maze	0.4917 ±0.0226	0.4101 ±0.0171	0.4522 ±0.0470	0.4836 ±0.0466	0.5267 ±0.0356	1.4555 ±0.0845	**0.3441** ±0.0288
Inclusions	0.6015 ±0.0297	0.4868 ±0.0666	0.5573 ±0.0275	0.8717 ±0.0508	0.5929 ±0.0381	1.4505 ±0.0715	**0.4261** ±0.0156
Discontinuous	0.4664 ±0.0569	0.3297 ±0.0168	0.4344 ±0.0510	0.6521 ±0.0416	0.4915 ±0.0097	1.3679 ±0.0841	**0.2890** ±0.0466
Joint	0.9757 ±0.3032	0.8013 ±0.1964	0.8646 ±0.2153	0.8792 ±0.0736	0.9449 ±0.1770	1.4074 ±0.0439	**0.7376** ±0.1034
FlowBench FPO (Tali et al., 2025)							
#Params	30.8 M	38.6 M	39.6 M	32.9 M	40.2 M	36.1 M	38.6 M
GFLOPs	468.2	553.7	606.1	464.6	686.0	77.1	481.2
vRMSE	0.3755 ±0.0029	0.3240 ±0.0265	0.3783 ±0.0254	0.3236 ±0.0098	0.3431 ±0.0353	0.3959 ±0.0065	**0.3177** ±0.0120
CO_2–water flow in porous media (Abdellatif et al., 2026)							
#Params	29.9 M	37.3 M	38.8 M	32.3 M	35.8 M	35.3 M	37.7 M
GFLOPs	2005.2	2175.5	2604.7	2002.8	3203.0	301.2	2067.0
vRMSE	0.7875 ±0.0039	0.7966 ±0.0072	0.7847 ±0.0030	**0.7785** ±0.0054	0.8027 ±0.0141	0.8142 ±0.0046	0.7893 ±0.0073

5 Discussion

5.1 What makes PAdaNorm conditioning effective?

We investigate how best to integrate PAdaNorm within the transformer block by varying three design axes (see Tab. 8 in App. B.3 for the full specification of each variant): (i) which attention axis is

Table 2: **`PAdaNorm` conditioning mode**: long-horizon validation vRMSE (↓, $T \in [21{:}60]$) on three acoustic scattering datasets from The Well (Ohana et al., 2024). We highlight the best metric in **bold** and underline the second best. The *Joint* column reports vRMSE averaged over the three datasets, each contributing 200 validation windows, for a single model trained on all three simultaneously. All entries are single runs (seed 1234); the three-seed results for spatial-only are in Tab. 1.

Conditioning	#Params	Maze	Inclusions	Discontinuous	Joint
Spatial-only	19.1 M	0.3716	**0.4094**	**0.2352**	0.8015
Temporal-only	19.1 M	**0.2996**	0.7522	0.3864	0.9520
Temporal (shared proj.)	19.1 M	0.3352	0.9115	0.4749	1.0266
Temporal + Spatial	23.1 M	0.3203	0.7745	0.3732	0.9325
Fully shared	15.6 M	0.3447	0.4819	0.3553	0.7899
Fully shared (spatial)	15.6 M	0.3522	0.5263	0.2657	0.8628
Fully shared (temporal)	15.6 M	0.3123	0.5745	0.3864	0.9823
Shared per-axis	16.9 M	0.3360	0.5525	0.3199	**0.6726**

modulated—spatial, temporal, or both; (ii) whether each block owns a dedicated projection or a single MLP is shared across all blocks; and (iii) when both axes are modulated, whether they receive axis-specific projections or share one. Tab. 2 reports the long-horizon vRMSE (single seed) for all eight variants on the acoustic scattering datasets.

No single variant dominates The best-performing variant depends on the dataset: temporal-only achieves the lowest error on Maze, spatial-only on Inclusions and Discontinuous, and shared per-axis on Joint. Spatial-only nonetheless achieves the lowest error on two settings (Inclusions and Discontinuous), more than any other variant, and is therefore the variant retained in our main comparison (Tab. 1).

Which attention axis to modulate Modulating a single attention axis (spatial-only or temporal-only, 19.1 M) generally matches or outperforms the dual-axis temporal+spatial variant (23.1 M), despite the latter using 21% more parameters. For instance, on Inclusions, spatial-only reaches 0.4094 versus 0.7745 for temporal+spatial. Furthermore, broadcasting a shared projection to both axes (temporal, shared proj.) underperforms both dedicated single-axis variants on three of the four settings, suggesting that forcing a single representation to serve both attention types dilutes the conditioning signal.

Parameter sharing across blocks Shared variants, which use a single conditioning MLP across all blocks (fully shared and shared per-axis), are competitive with per-block variants despite a lower parameter count (15.6–16.9 M vs. 19.1 M for spatial-only). On individual datasets, they often match per-block performance, e.g., fully shared (temporal) achieves 0.3123 on Maze, close to temporal-only's 0.2996. Shared variants even surpass the per-block variant on Joint: shared per-axis reaches 0.6726 and fully shared reaches 0.7899, compared to 0.8015 for spatial-only, with 12% and 18% fewer parameters. This suggests that per-block specialization is not strictly necessary and that a lightweight, shared conditioning path can be an efficient alternative.

5.2 Overlapping patches

By default, the conditioning fields are partitioned into non-overlapping patches aligned one-to-one with the state patch grid, so each conditioning token captures only the geometry local to its corresponding state patch. To allow each conditioning token to incorporate context from neighboring patches, we consider three alternatives displayed in Fig. 5: sliding-window MLP, convolutional encoder and coarse patches. We provide a detailed description of each in App. C.2. The sliding-window MLP includes conditioning patches that overlap their neighbors via a sliding window of size $k > P$, so that a shared MLP sees context beyond the corresponding state patch. The convolutional encoder includes a stack of convolutional layers that process the conditioning fields at full spatial resolution, followed by a strided convolution that downsamples to the patch grid, allowing the receptive field to grow with depth. Finally, Coarse conditioning patches involve partitioning the conditioning fields into patches of size $mP \times mP$, each covering $m \times m$ state patches, and the resulting conditioning vector is broadcast identically to all covered tokens.

Table 3: Effect of conditioning patch strategy for spatial-only PAdaNorm on validation vRMSE (↓) across three acoustic scattering datasets (60-step autoregressive rollout). We compare the standard (base) patching against three alternatives: sliding-window overlap, convolutional overlap, and coarse 2×. All metrics are reported at the checkpoint with the best $T\in[21{:}60]$ score. We highlight the best metric in **bold** and underline the second best. All entries are single runs (seed 1234).

Patch strategy	#Params	Maze		Inclusions		Discontinuous	
		$T\in[21{:}60]$	Overall	$T\in[21{:}60]$	Overall	$T\in[21{:}60]$	Overall
Base	19.1 M	0.3716	0.2938	0.4094	0.3054	0.2352	0.1732
Sliding-window overlap	19.1 M	0.3770	0.2982	**0.3461**	**0.2629**	**0.2291**	**0.1688**
Convolutional overlap*	20.8 M	**0.2927**	**0.2236**	0.4076	0.3037	0.2511	0.1852
Coarse 2×	19.3 M	0.4457	0.3374	0.5770	0.4311	0.3566	0.2604

We evaluate all three strategies on the acoustic scattering datasets with the spatial-only PAdaNorm conditioning (Tab. 3). The sliding-window overlap improves over the non-overlapping baseline on two of three datasets at no additional parameter cost, with the largest gain on Inclusions (-15% in long-horizon vRMSE). The convolutional encoder achieves the best result on Maze (-21%), is on par with the baseline on Inclusions, and is worse on Discontinuous ($+7\%$), despite a moderate increase in parameters. Coarse 2× patches degrade performance on all three datasets. These results are consistent with the physics of acoustic scattering, where the most important features are material interfaces, at which waves are reflected and diffracted. When such an interface falls on a patch edge, non-overlapping patches split it across two tokens, each missing the full local geometry. Overlapping strategies mitigate this by providing each conditioning token with context from neighboring patches; conversely, coarsening trades spatial resolution for receptive field size, losing the fine-grained geometric detail that drives accurate scattering predictions.

6 Conclusion

We investigated how explicitly conditioning on spatial inputs, such as material properties and obstacle geometry, affects transformer-based PDE surrogates. We introduced Patchwise Adaptive Normalization (PAdaNorm) for exogenous fields that influence the evolving state. Through a shared encoder and block-specific projections, we map these fields to spatially aligned scale, shift, and gate parameters that modulate the transformer's intermediate representations.

Across three physical domains, this conditioning approach improves or matches prediction accuracy while reducing rollout computation relative to similarly sized baselines. On acoustic scattering, we obtain reductions of up to 16% in mean long-horizon variance-scaled RMSE compared with parameter-matched baselines that concatenate conditioning fields with the input state. Across all benchmarks, we obtain improved or competitive accuracy with fewer rollout FLOPs than the wider and deeper models or the cross-attention baselines. PAdaNorm also better maintains hard physical constraints over long rollouts, such as preventing flow into impenetrable solid regions in porous media. Further, since the conditioning fields considered are static, modulation parameters can be cached once and reused throughout inference, amortizing the associated computational cost. Ablations further show that preserving distinct conditioning vectors at the patch resolution is beneficial, while additional spatial context and modulation complexity have dataset-dependent effects. Taken together, the spatial conditioning provided by PAdaNorm is an effective use of model capacity, improving prediction accuracy with fewer forward-pass FLOPs than parameter-matched baselines (App. B.5).

Limitations and future work Our evaluation is restricted to two-dimensional PDEs and a single backbone architecture (Walrus); validating PAdaNorm on three-dimensional domains and other transformer-based surrogates remains open. All conditioning fields considered in this work are static; extending PAdaNorm to time-varying exogenous fields, where the conditioning must be recomputed at each rollout step, is a natural next step. The optimal PAdaNorm variant (spatial-only vs. temporal-only) is dataset-dependent, and we currently lack a principled criterion for selecting the modulation axis a priori. Finally, while we empirically observe that PAdaNorm lowers long-horizon errors on most of the datasets considered and better maintains hard physical constraints over long rollouts, a theoretical understanding of why spatially localized modulation has these effects would further strengthen the approach.

AI USE STATEMENT

We used generative AI tools to (i) help with LaTeX and Ti*k*Z formatting, (ii) proofread and polish the manuscript, (iii) implement some of the ablation methods, (iv) design and give feedback on experiments, (v) assist in interpreting results and (vi) AI agents were used to launch and monitor experiments. We did not use generative AI tools to clean or reformat datasets, support qualitative or thematic data analysis, generate synthetic datasets, or propose or refine hypotheses. Developing theoretical models or conceptual frameworks, formulating mathematical claims or providing critical ingredients for proving them, and writing proofs are not applicable to this work. All AI-assisted code was thoroughly reviewed by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We describe the Walrus-inspired backbone and the `PAdaNorm` conditioning architecture in Sec. 3, with the spatial and temporal blocks detailed in App. A.1 and every conditioning variant in App. B.3. Complete per-model configurations for every dataset (architecture, preprocessing, normalization, optimization and evaluation settings) are listed in App. B.4, and the GNOT baselines in App. B.3.1. App. B.5 gives the FLOP-counting convention and the derivations, checked against measured counts. The acoustic scattering (The Well), FlowBench and CO_2–water datasets are publicly available. Results in the main comparison (Tab. 1) are averaged over three seeds (1234, 21, 67). All metrics are computed on the validation windows, which are also used to select the checkpoint with the best $T{\in}[21{:}60]$ score; we do not use a separate test split. Our implementation depends on proprietary training infrastructure and cannot be released. To support reimplementation, we give the exact equations of every conditioning variant, the complete hyperparameter configurations and the FLOP derivations referenced above.

REFERENCES

Alhasan Abdellatif, Hannah P. Menke, Julien Maes, Ahmed H. Elsheikh, and Florian Doster. A benchmark dataset for machine learning surrogates of pore-scale CO_2-water interaction. *Scientific Data*, 13:621, 2026. doi: 10.1038/s41597-025-05794-z.

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. doi: 10.52202/079017-0793.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. arXiv preprint arXiv:1607.06450, 2016.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, volume 28, pp. 1171–1179. Curran Associates, Inc., 2015.

Johannes Brandstetter, Daniel E. Worrall, and Max Welling. Message passing neural PDE solvers. In *International Conference on Learning Representations*, 2022.

Shoufa Chen, Mengmeng Xu, Jiawei Ren, Yuren Cong, Sen He, Yanping Xie, Animesh Sinha, Ping Luo, Tao Xiang, and Juan-Manuel Perez-Rua. GenTron: Diffusion transformers for image and video generation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6441–6451, 2024.

Prafulla Dhariwal and Alex Nichol. Diffusion models beat GANs on image synthesis. In *Advances in Neural Information Processing Systems*, volume 34, pp. 8780–8794, 2021.

Vincent Dumoulin, Ethan Perez, Nathan Schucher, Florian Strub, Harm de Vries, Aaron Courville, and Yoshua Bengio. Feature-wise transformations. *Distill*, 3(7):e11, 2018. doi: 10.23915/distill.00011.

Zhongkai Hao, Zhengyi Wang, Hang Su, Chengyang Ying, Yinpeng Dong, Songming Liu, Ze Cheng, Jian Song, and Jun Zhu. GNOT: A general neural operator transformer for operator learning. In

International Conference on Machine Learning, volume 202 of *Proceedings of Machine Learning Research*, pp. 12556–12569, 2023.

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*, volume 235 of *Proceedings of Machine Learning Research*, pp. 17616–17635, 2024.

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*, volume 37, 2024.

Benjamin Holzschuh, Qiang Liu, Georg Kohl, and Nils Thuerey. PDE-Transformer: Efficient and versatile transformers for physics simulations. In *International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 23562–23602, 2025.

Benjamin Holzschuh, Georg Kohl, Florian Redinger, and Nils Thuerey. P3D: Highly scalable 3D neural surrogates for physics simulations with global context. In *International Conference on Learning Representations*, 2026.

Felix Koehler, Simon Niedermayr, Rüdiger Westermann, and Nils Thuerey. APEBench: A benchmark for autoregressive neural emulators of PDEs. In *Advances in Neural Information Processing Systems*, volume 37, pp. 120252–120310. Curran Associates, Inc., 2024. doi: 10.52202/079017-3822.

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.

Phillip Lippe, Bastiaan S. Veeling, Paris Perdikaris, Richard E. Turner, and Johannes Brandstetter. PDE-Refiner: Achieving accurate long rollouts with neural PDE solvers. In *Advances in Neural Information Processing Systems*, volume 36, pp. 67398–67433, 2023.

Qibang Liu, Weiheng Zhong, Hadi Meidani, Diab Abueidda, Seid Koric, and Philippe Geubelle. Geometry-informed neural operator transformer for partial differential equations on arbitrary geometries. *Computer Methods in Applied Mechanics and Engineering*, 451:118668, 2026. doi: 10.1016/j.cma.2025.118668.

Michael McCabe, Bruno Régaldo-Saint Blancard, Liam Holden Parker, Ruben Ohana, Miles Cranmer, Alberto Bietti, Michael Eickenberg, Siavash Golkar, Géraud Krawezik, François Lanusse, Mariel Pettee, Tiberiu Tesileanu, Kyunghyun Cho, and Shirley Ho. Multiple physics pretraining for spatiotemporal surrogate models. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-3791.

Michael McCabe, Payel Mukhopadhyay, Tanya Marwah, Bruno Regaldo-Saint Blancard, Francois Rozet, Cristiana Diaconu, Lucas Meyer, Kaze W. K. Wong, Hadi Sotoudeh, Alberto Bietti, Irina Espejo, Rio Fear, Siavash Golkar, Tom Hehir, Keiya Hirashima, Geraud Krawezik, Francois Lanusse, Rudy Morel, Ruben Ohana, Liam Parker, Mariel Pettee, Jeff Shen, Kyunghyun Cho, Miles Cranmer, and Shirley Ho. Walrus: A cross-domain foundation model for continuum dynamics. In *International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, 2026.

Ruben Ohana, Michael McCabe, Lucas Meyer, Rudy Morel, Fruzsina J. Agocs, Miguel Beneitez, Marsha Berger, Blakesley Burkhart, Stuart B. Dalziel, Drummond B. Fielding, Daniel Fortunato, Jared A. Goldberg, Keiya Hirashima, Yan-Fei Jiang, Rich R. Kerswell, Suryanarayana Maddu, Jonah Miller, Payel Mukhopadhyay, Stefan S. Nixon, Jeff Shen, Romain Watteaux, Bruno Régaldo-Saint Blancard, François Rozet, Liam Parker, Miles Cranmer, and Shirley Ho. The Well: a large-scale collection of diverse physics simulations for machine learning. In *Advances in Neural Information Processing Systems*, volume 37, pp. 44989–45037, 2024.

Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2337–2346, 2019.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *International Conference on Computer Vision*, pp. 4195–4205, 2023.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. FiLM: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. doi: 10.1609/aaai.v32i1.11671.

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.

Sucheng Ren, Qihang Yu, Ju He, Xiaohui Shen, Alan Yuille, and Liang-Chieh Chen. FlowAR: Scale-wise autoregressive image generation meets flow matching. In *International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 51489–51502, 2025.

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*, pp. 8459–8468. PMLR, 2020.

Ronak Tali, Ali Rabeh, Cheng-Hau Yang, Mehdi Shadkhah, Samundra Karki, Abhisek Upadhyaya, Suriya Dhakshinamoorthy, Marjan Saadati, Soumik Sarkar, Adarsh Krishnamurthy, Chinmay Hegde, Aditya Balu, and Baskar Ganapathysubramanian. FlowBench: A large scale benchmark for flow simulation over complex geometries. *Journal of Data-centric Machine Learning Research*, 2: 1–35, 2025.

Florian Wiesner, Zoë J. Gray, Matthias Wessling, and Stephen Baek. Towards a physics foundation model. arXiv preprint arXiv:2509.13805, 2026.

Yongsheng Yu, Wei Xiong, Weili Nie, Yichen Sheng, Shiqiu Liu, and Jiebo Luo. PixelDiT: Pixel diffusion transformers for image generation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14273–14282, 2026.

A Model details

A.1 Walrus Transformer

We detail our Walrus-inspired architecture here, which retains the spatial and temporal attention blocks of the original Walrus backbone (McCabe et al., 2026).

Temporal attention In the temporal attention block, each spatial patch attends only to patches at the same spatial location across the timesteps within the input window. Formally, given an input spatiotemporal feature tensor $\mathbf{X} \in \mathbb{R}^{B\times T_{\text{in}}\times N_x\times N_y\times D}$, where B is the batch size, T_{in} is the number of context timesteps, and D is the embedding dimension, the temporal block performs multi-head self-attention (MHSA) independently across the time axis for each spatial token. For a patch position $s \in [1, N_x] \times [1, N_y]$, let $\mathbf{X}_s \in \mathbb{R}^{B\times T_{\text{in}}\times D}$ denote the sub-tensor of $\mathbf{X}$ at that position and $\tilde{\mathbf{X}}_s = \texttt{RMSGroupNorm}(\mathbf{X}_s)$, where $\texttt{RMSGroupNorm}$ partitions the D channels into N_H groups (N_H being the number of attention heads), applies RMS normalization within each group and rescales by a learned per-channel gain. The queries, keys, and values are mapped via projection weights $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v \in \mathbb{R}^{D\times D}$, so that the updated representation $\mathbf{X}'_s$ reads:

$$\mathbf{X}'_s = \mathbf{X}_s + \texttt{Softmax}\left(\frac{(\tilde{\mathbf{X}}_s\mathbf{W}_q)(\tilde{\mathbf{X}}_s\mathbf{W}_k)^\top}{\sqrt{d_k}}\right)(\tilde{\mathbf{X}}_s\mathbf{W}_v)\mathbf{W}_o, \tag{7}$$

where $\mathbf{W}_o \in \mathbb{R}^{D\times D}$ is the output projection matrix and d_k is the per-head scaling dimension; per-head query–key normalization and the 1D temporal RoPE are applied to the queries and keys, and are omitted for simplicity of notation. As noted in Sec. 3, this attention is unmasked: all T_{in} frames of the input window are observed, so the block acts as a bidirectional encoder over the context, while the prediction itself remains causal by construction since the target frame lies outside the window. The temporal sublayer carries no feed-forward block, as in the original Walrus implementation. The full intermediate tensor $\mathbf{X}' \in \mathbb{R}^{B\times T_{\text{in}}\times N_x\times N_y\times D}$ is reconstructed by stacking $\mathbf{X}'_s$ over spatial positions.

Spatial attention Following the temporal update, the intermediate tensor $\mathbf{X}'$ is forwarded to the spatial attention block, which allows each token to attend to all other positions at the same time instant. Unlike the temporal sublayer, this block carries out the layer's feed-forward computation. It does so in *parallel* with the attention rather than in sequence: following the spatial attention design of Walrus (McCabe et al., 2026), a single normalization feeds one fused projection that produces the attention queries, keys, and values together with the gate and up projections of a SwiGLU feed-forward branch, and the two branch outputs are summed into a single residual.

For a fixed timestep $t \in [1, T_{\text{in}}]$, let $\mathbf{X}'_t = \mathbf{X}'_{:,t,:,:} \in \mathbb{R}^{B\times N_x\times N_y\times D}$ and $\tilde{\mathbf{X}}'_t = \texttt{RMSGroupNorm}(\mathbf{X}'_t)$. The fused projection $\mathbf{W}_{\text{fused}} \in \mathbb{R}^{D\times(3D+2D_{\text{ff}})}$ yields

$$[\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{G}, \mathbf{U}] = \tilde{\mathbf{X}}'_t\mathbf{W}_{\text{fused}}, \tag{8}$$

with $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{B\times N_xN_y\times D}$ and $\mathbf{G}, \mathbf{U} \in \mathbb{R}^{B\times N_xN_y\times D_{\text{ff}}}$, where D_{ff} is the feed-forward width. Omitting the 2D RoPE applied to $\mathbf{Q}$ and $\mathbf{K}$, the output representation $\mathbf{X}''_t$ reads:

$$\mathbf{X}''_t = \mathbf{X}'_t + \underbrace{\texttt{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}\mathbf{W}'_o}_{\text{attention branch}} + \underbrace{(\texttt{SiLU}(\mathbf{G})\odot\mathbf{U})\,\mathbf{W}_{\text{out}}}_{\text{feed-forward branch}}, \tag{9}$$

where $\mathbf{W}'_o \in \mathbb{R}^{D\times D}$ and $\mathbf{W}_{\text{out}} \in \mathbb{R}^{D_{\text{ff}}\times D}$, and $\odot$ denotes the element-wise product.

B Experiments

B.1 Results

Tab. 4 reports the full acoustic results at all three horizons, and Tabs. 5 and 6 those for FlowBench and CO_2.

Table 4: Best validation vRMSE (↓) across conditioning modes on three acoustic scattering datasets from The Well (Ohana et al., 2024) (60-step autoregressive rollout). All models share similar hyperparameters where relevant. Rows that state a spread are the mean ± sample standard deviation over three seeds (1234, 21, 67), each interval read at that seed's best $T \in [21{:}60]$ checkpoint; rows without one are marked as single-seed and are not directly comparable to them. We highlight the best metric in **bold** and underline the second best. GFLOPs is the batch-1 forward pass, as in Tab. 12. †Parameter-matched In-context controls use the same total parameter count as the `PAdaNorm` temporal+spatial variants (∼23 M). ‡Parameter-matched In-context controls targeting the per-block `PAdaNorm` parameter count (∼19 M). The *Joint* column reports vRMSE averaged over the three datasets, each contributing 200 validation windows, for a single model trained on all three simultaneously.

Conditioning	#Params	GFLOPs	Maze			Inclusions			Discontinuous			Joint		
			$T{=}1$	$T{\in}[1{:}20]$	$T{\in}[21{:}60]$	$T{=}1$	$T{\in}[1{:}20]$	$T{\in}[21{:}60]$	$T{=}1$	$T{\in}[1{:}20]$	$T{\in}[21{:}60]$	$T{=}1$	$T{\in}[1{:}20]$	$T{\in}[21{:}60]$
Baselines														
In-context	13.3 M	113.5	0.0267 ±0.0021	0.1165 ±0.0129	0.4917 ±0.0226	0.0179 ±0.0009	0.1264 ±0.0061	0.6015 ±0.0297	0.0084 ±0.0001	0.0807 ±0.0058	0.4664 ±0.0569	0.0248 ±0.0031	0.1629 ±0.0275	0.9757 ±0.3032
In-context† (wide) (single seed)	23.9 M	184.8	0.0250	0.1089	0.4566	**0.0140**	0.1119	0.5828	0.0086	0.0865	0.4255	**0.0179**	0.1251	0.6973
In-context† (deep) (single seed)	23.9 M	203.8	0.0245	0.1022	0.4472	0.0158	0.1172	0.6818	0.0076	0.0824	0.4855	0.0187	0.1263	0.8152
In-context‡ (wide)	18.0 M	138.9	**0.0232** ±0.0004	0.0987 ±0.0050	0.4101 ±0.0171	0.0141 ±0.0008	**0.0976** ±0.0135	0.4868 ±0.0666	**0.0069** ±0.0005	0.0583 ±0.0019	0.3297 ±0.0168	0.0194 ±0.0014	0.1399 ±0.0074	0.8013 ±0.1964
In-context‡ (deep)	18.6 M	158.6	0.0292 ±0.0012	0.1178 ±0.0102	0.4522 ±0.0470	0.0166 ±0.0005	0.1111 ±0.0106	0.5573 ±0.0275	0.0081 ±0.0010	0.0781 ±0.0106	0.4344 ±0.0510	0.0215 ±0.0021	0.1571 ±0.0446	0.8646 ±0.2153
AdaNorm (global)	15.5 M	113.3	0.0328 ±0.0030	0.1856 ±0.0120	0.4836 ±0.0466	0.0185 ±0.0005	0.2043 ±0.0012	0.8717 ±0.0508	0.0080 ±0.0003	0.1127 ±0.0116	0.6521 ±0.0416	0.0241 ±0.0033	0.2031 ±0.0100	0.8792 ±0.0736
Cross-attention	18.5 M	179.7	0.0249 ±0.0009	0.1129 ±0.0103	0.5267 ±0.0356	0.0169 ±0.0010	0.1207 ±0.0050	0.5929 ±0.0381	0.0078 ±0.0006	0.0799 ±0.0056	0.4915 ±0.0097	0.0220 ±0.0009	0.1541 ±0.0146	0.9449 ±0.1770
GNOT	18.8 M	41.2	0.2690 ±0.0776	1.0853 ±0.0396	1.4555 ±0.0845	0.0943 ±0.0289	0.9829 ±0.1135	1.4505 ±0.0715	0.1107 ±0.0432	0.9758 ±0.0907	1.3679 ±0.0841	0.2207 ±0.0871	1.0577 ±0.1102	1.4074 ±0.0439
`PAdaNorm` *conditioning* (ours)														
Spatial-only	19.1 M	125.2	0.0300 ±0.0063	0.1118 ±0.0230	0.3441 ±0.0288	0.0176 ±0.0008	0.1064 ±0.0116	**0.4261** ±0.0156	0.0077 ±0.0003	0.0579 ±0.0078	0.2890 ±0.0466	0.0242 ±0.0028	**0.1223** ±0.0055	0.7376 ±0.1034
Temporal-only (single seed)	19.1 M	125.2	0.0270	0.0942	**0.2996**	0.0172	0.1238	0.7522	0.0076	0.0700	0.3864	0.0218	0.1249	0.9520
Temporal (shared proj.) (single seed)	19.1 M	125.2	0.0253	0.1000	0.3352	0.0247	0.2026	0.9115	0.0074	0.0740	0.4749	0.0573	0.2881	1.0266
Temporal + Spatial (single seed)	23.1 M	133.3	0.0251	0.1003	0.3203	0.0179	0.1215	0.7745	0.0074	0.0758	0.3732	0.0530	0.2317	0.9325
Fully shared (single seed)	15.6 M	118.0	0.0260	**0.0898**	0.3447	0.0169	0.1030	0.4819	0.0080	0.0663	0.3553	0.0254	0.1558	0.7899
Fully shared (spatial) (single seed)	15.6 M	118.0	0.0698	0.1555	0.3522	0.0169	0.0994	0.5263	0.0083	**0.0570**	**0.2657**	0.0218	0.1253	0.8628
Fully shared (temporal) (single seed)	15.6 M	118.0	0.0488	0.1226	0.3123	0.0174	0.1118	0.5745	0.0072	0.0756	0.3864	0.0342	0.1842	0.9823
Shared per-axis (single seed)	16.9 M	120.7	0.0271	0.1000	0.3360	0.0166	0.1101	0.5525	0.0085	0.0692	0.3199	0.0224	0.1239	**0.6726**

Table 5: Best validation vRMSE (↓) for baselines and `PAdaNorm` spatial-only on FlowBench FPO (Tali et al., 2025) (2D Navier–Stokes, 1024×256, 60-step autoregressive rollout). ‡Parameter-matched In-context controls targeting the `PAdaNorm` parameter count (∼38.6 M). Each entry is the mean ± sample standard deviation over three seeds (1234, 21, 67). Per seed the three intervals are read at that seed's best $T \in [21{:}60]$ checkpoint, so each row describes one model per seed. GFLOPs is the batch-1 forward pass, reproduced from Tab. 12. We highlight the best mean in **bold** and underline the second best.

Conditioning	#Params	GFLOPs	$T=1$	$T \in [1{:}20]$	$T \in [21{:}60]$
Baselines					
In-context	30.8 M	468.2	0.0014 ±0.0000	0.0455 ±0.0022	0.3755 ±0.0029
In-context‡ (wide)	38.6 M	553.7	0.0013 ±0.0000	0.0417 ±0.0025	0.3240 ±0.0265
In-context‡ (deep)	39.6 M	606.1	0.0019 ±0.0009	0.0527 ±0.0134	0.3783 ±0.0254
AdaNorm (global)	32.9 M	464.6	0.0014 ±0.0001	<u>0.0402</u> ±0.0017	<u>0.3236</u> ±0.0098
Cross-attention	40.2 M	686.0	<u>0.0013</u> ±0.0001	0.0435 ±0.0039	0.3431 ±0.0353
GNOT	36.1 M	77.1	0.0026 ±0.0001	0.0651 ±0.0017	0.3959 ±0.0065
`PAdaNorm` conditioning (ours)					
Spatial-only	38.6 M	481.2	**0.0012** ±0.0001	**0.0392** ±0.0036	**0.3177** ±0.0120

Table 6: Best validation vRMSE (↓) for baselines and `PAdaNorm` spatial-only on CO_2–water flow in porous media (Abdellatif et al., 2026) (512×512, $T_{\text{in}}=4$, 60-step autoregressive rollout). ‡Parameter-matched In-context controls targeting the `PAdaNorm` parameter count (∼37.7 M). Each entry is the mean ± sample standard deviation over three seeds (1234, 21, 67). Per seed the three intervals are read at that seed's best $T \in [21{:}60]$ checkpoint, so each row describes one model per seed. GFLOPs is the batch-1 forward pass, reproduced from Tab. 12. We highlight the best mean in **bold** and underline the second best.

Conditioning	#Params	GFLOPs	$T=1$	$T \in [1{:}20]$	$T \in [21{:}60]$
Baselines					
In-context	29.9 M	2005.2	0.2652 ±0.0029	0.4703 ±0.0064	0.7875 ±0.0039
In-context‡ (wide)	37.3 M	2175.5	0.2607 ±0.0036	0.4760 ±0.0053	0.7966 ±0.0072
In-context‡ (deep)	38.8 M	2604.7	**0.2571** ±0.0043	<u>0.4682</u> ±0.0035	<u>0.7847</u> ±0.0030
AdaNorm (global)	32.3 M	2002.8	0.2605 ±0.0023	0.4697 ±0.0054	**0.7785** ±0.0054
Cross-attention	35.8 M	3203.0	0.2644 ±0.0013	0.4781 ±0.0135	0.8027 ±0.0141
GNOT	35.3 M	301.2	0.3034 ±0.0009	0.5214 ±0.0029	0.8142 ±0.0046
`PAdaNorm` conditioning (ours)					
Spatial-only	37.7 M	2067.0	<u>0.2591</u> ±0.0027	**0.4666** ±0.0145	0.7893 ±0.0073

B.2 Qualitative rollouts

We complement the aggregate error metrics with selected trajectories to illustrate how prediction errors develop spatially. In the FlowBench example (Fig. 2), both `PAdaNorm` and the parameter-matched wide In-context baseline initially reproduce the pressure pattern downstream of the obstacle. At later steps, we observe differences in the positions and shapes of these structures, together with distortions near the obstacle. In the acoustic example (Fig. 3), we observe discrepancies in

Table 7: Mean absolute error on *rock* pixels (↓, z-score normalized units) on CO_2–water flow in porous media, averaged over $T \in [21{:}60]$. All fields must be zero within the rock pixels. Each entry is the mean ± sample standard deviation over three seeds (1234, 21, 67), each read at its best $T \in [21{:}60]$ checkpoint, as in Tab. 6. ‡Parameter-matched In-context controls. We highlight the best mean in **bold** and underline the second best.

Conditioning	Saturation	Pressure	U_x	U_y
Baselines				
In-context	0.083 ±0.002	0.157 ±0.023	0.204 ±0.015	0.224 ±0.017
In-context‡ (wide)	0.089 ±0.009	0.224 ±0.073	0.260 ±0.026	0.253 ±0.024
In-context‡ (deep)	0.075 ±0.006	<u>0.155</u> ±0.007	0.240 ±0.043	0.239 ±0.027
AdaNorm (global)	<u>0.073</u> ±0.013	0.174 ±0.010	<u>0.182</u> ±0.015	<u>0.192</u> ±0.013
Cross-attention	0.091 ±0.006	0.193 ±0.018	0.216 ±0.018	0.235 ±0.014
GNOT	0.101 ±0.023	0.187 ±0.049	0.227 ±0.026	0.229 ±0.011
PAdaNorm conditioning (ours)				
Spatial-only	**0.062** ±0.002	**0.114** ±0.002	**0.142** ±0.008	**0.145** ±0.004

the scattered wave pattern and grid-aligned artifacts as the rollout progresses. On CO_2 (Fig. 4), both models initially reproduce the main flow paths, but the depth-matched In-context baseline progressively predicts spurious velocity inside the rock, whereas PAdaNorm does so noticeably less. Neither model reproduces a large invasion region that develops later in the trajectory, illustrating how an early error in the CO_2 front propagates to all future rollout steps.

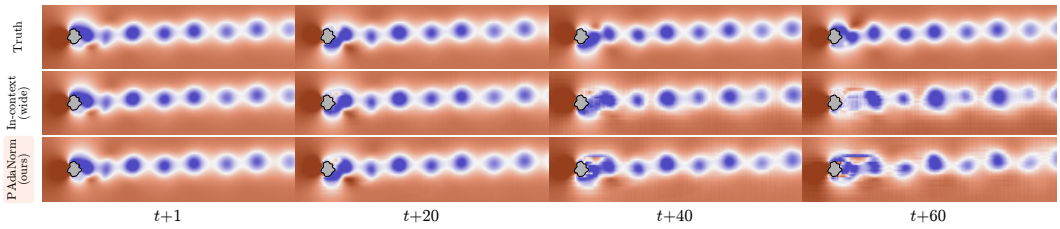


Figure 2: Autoregressive rollout on FlowBench FPO (pressure channel, Reynolds number $\mathrm{Re}=364$). Rows: ground truth, the parameter-matched wide In-context baseline ($D=512$, 7 blocks), and PAdaNorm spatial-only ($D=384$, 10 blocks); each rollout is generated by the seed-1234 checkpoint from Tab. 1. The obstacle is overlaid in gray.



Figure 3: Typical autoregressive rollout on acoustic scattering with inclusions (pressure). Rows: ground truth, the parameter-matched wide In-context baseline (D=384, 6 blocks), the strongest In-context baseline on this dataset in Tab. 1, and `PAdaNorm` spatial-only (D=256, 10 blocks); each rollout is generated by a seed-21 checkpoint from the runs whose metrics enter that table. The inclusions are outlined in gray. The window is the one whose `PAdaNorm`-vs-baseline error gap is closest to the median over the candidates considered, so it is neither a best- nor a worst-case draw.

B.3 METHODS

Tab. 4 compares different approaches to conditioning on the one-way coupled fields; we describe each of them here.

Baseline methods:

1. **In-context**: The conditioning fields are concatenated to the state as additional input channels before patch embedding, hence the transformer must infer the effect of the conditioning fields from context alone.
2. **In-context (wide/deep)**: Parameter-matched models that raise the capacity of the In-context baseline. *Wide* has an increased embedding dimension, *deep* a larger number of blocks. † targets the parameter count of the spatial and temporal `PAdaNorm` variant. ‡ targets the spatial-only `PAdaNorm` variant.
3. **AdaNorm (global)**: The conditioning embedding here is obtained from a single globally average-pooled conditioning vector, which a single MLP, shared across all blocks, maps to shift, scale, and gate parameters for both attention sublayers. This provides spatially uniform modulation: the same conditioning signal is broadcast to every token regardless of its spatial position.
4. **GNOT**: see App. B.3.1 for extensive details on our implementation of GNOT for the present setting.

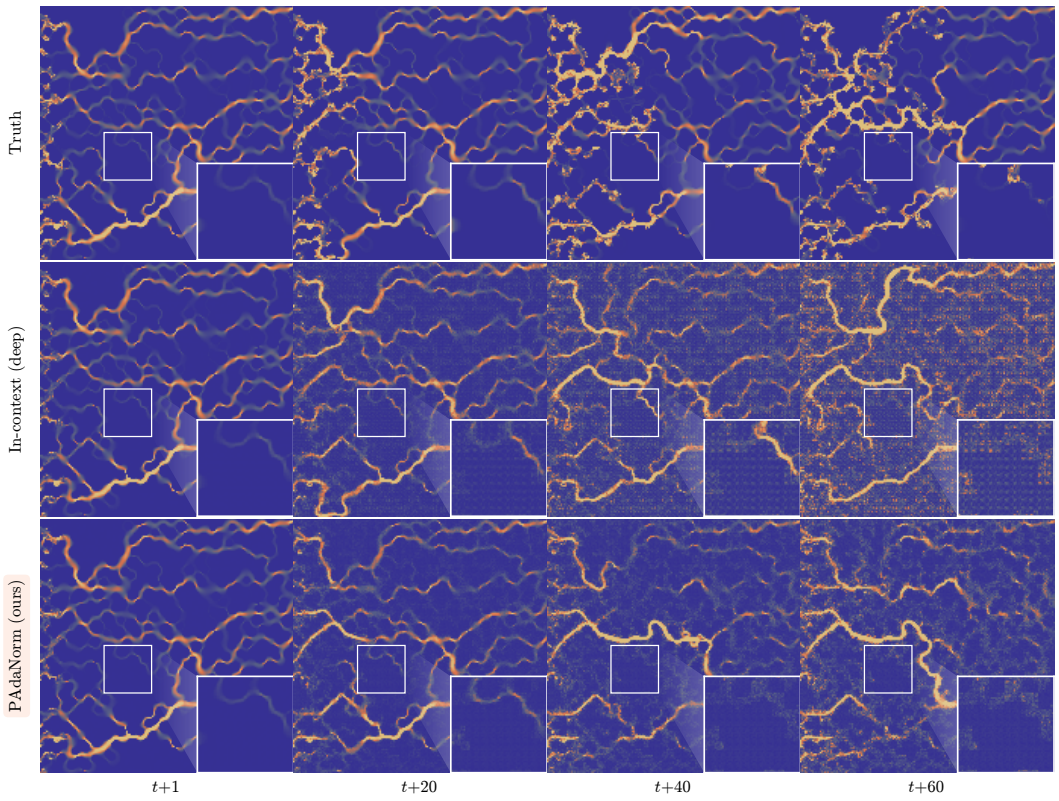


Figure 4: Typical autoregressive rollout on CO_2–water flow in porous media (Abdellatif et al., 2026), showing the velocity magnitude $\|\mathbf{u}\|$. The inset in the bottom-right corner of each panel magnifies the region outlined in white. The rows show the ground truth, In-context (deep) — the depth-matched parameter control — and `PAdaNorm` spatial-only. Throughout the rollout, `PAdaNorm` enforces the hard constraint of not having gas or water permeating the rock cells better than the parameter-matched In-context baseline, consistent with its lower error on rock pixels (Tab. 7).

All `PAdaNorm` variants partition the conditioning fields into patches aligned with the state patches and produce a spatially varying conditioning vector per patch position. A shared MLP maps each conditioning patch to a conditioning embedding, which is then projected into the adaptive normalization modulation parameters (shift, scale, gate) that are applied to the normalization and residual connection along the specified attention axis. Tab. 8 describes each variant along the following components:

- *Spatial modulation*: whether the spatial attention sublayer is modulated by `PAdaNorm`.
- *Temporal modulation*: whether the temporal attention sublayer is modulated by `PAdaNorm`.
- *Axis-specific*: temporal and spatial attention receive distinct modulation parameters via separate projections; otherwise, a single projection output is broadcast to both axes.
- *Block-specific*: each transformer block owns its own projection; otherwise, a single MLP is shared across all blocks.

B.3.1 GNOT Baseline

We re-implement the General Neural Operator Transformer (GNOT) (Hao et al., 2023) as a baseline, adapting it to our 2D PDE surrogate setting. Below, we describe the key implementation choices and how they relate to the original paper.

Tokenization GNOT operates on a set of query points rather than on a regular pixel grid. To enable a fair comparison with our patch-based transformer, we tokenize the input field using the same

Table 8: `PAdaNorm` conditioning variants.

Conditioning	Spatial modulation	Temporal modulation	Axis-specific	Block-specific
Spatial-only	✓	✗	–	✓
Temporal-only	✗	✓	–	✓
Temporal (shared proj.)	✓	✓	✗	✓
Temporal + Spatial	✓	✓	✓	✓
Fully shared	✓	✓	✗	✗
Fully shared (spatial)	✓	✗	–	✗
Fully shared (temporal)	✗	✓	–	✗
Shared per-axis	✓	✓	✓	✗

`PatchEmbed`/`PatchUnembed` modules as our main model. 2D grid coordinates (normalized to $[0, 1]$) are concatenated to each patch token and linearly projected back to the hidden dimension. The conditioning fields are embedded using the same `PatchEmbed` module as our main model, producing per-patch branch tokens for cross-attention conditioning.

Temporal handling Following common practice in neural-operator surrogates (Li et al., 2021), we channel-concatenate the T_{in} input timesteps, reshaping $(B, T_{\text{in}}, C, H, W)$ to $(B, T_{\text{in}} \cdot C, H, W)$ before tokenization. This avoids introducing a temporal attention axis absent from the original GNOT design.

Conditioning on geometry GNOT natively conditions on heterogeneous inputs through cross-attention with separate key/value projections per input type. We use this mechanism directly: the embedded conditioning fields are provided as a single-branch input to `LinearCrossAttention`. The conditioning branch tokens share the same patch grid as the state tokens.

Attention We use GNOT's linear attention formulation, which applies softmax normalization along the *feature* dimension of Q and K rather than the token dimension:

$$\text{Attn}(Q, K, V) = \sigma_d(Q) \left[\sigma_d(K)^\top V\right], \tag{10}$$

where σ_d denotes row-wise softmax over the head dimension d. This yields $O(Nd^2)$ complexity instead of the $O(N^2 d)$ of standard attention.

Transformer block Each GNOT block follows the structure:

$$\mathbf{x} \leftarrow \mathbf{x} + \text{CrossAttn}\big(\text{LN}(\mathbf{x}),\, \text{LN}(\mathbf{y})\big), \tag{11}$$

$$\mathbf{x} \leftarrow \mathbf{x} + \text{FFN}_1\big(\text{LN}(\mathbf{x})\big), \tag{12}$$

$$\mathbf{x} \leftarrow \mathbf{x} + \text{SelfAttn}\big(\text{LN}(\mathbf{x})\big), \tag{13}$$

$$\mathbf{x} \leftarrow \mathbf{x} + \text{FFN}_2\big(\text{LN}(\mathbf{x})\big), \tag{14}$$

where each FFN is a residual MLP with GELU activations and $4\times$ expansion ratio.

Mixture of Experts The original GNOT supports geometric-gating MoE, where each FFN is replaced by K expert MLPs gated by the spatial coordinates: $\text{gate}(\mathbf{x}_{\text{pos}}) = \text{softmax}\big(W \mathbf{x}_{\text{pos}}\big)$. In our experiments, we use $K = 1$, matching the non-MoE ablation in the original paper and found no consistent improvement when using MoE's.

Positional encoding GNOT supports optional Fourier positional features for the spatial coordinates. We implement this as a configurable option (`horiz_fourier_dim`; default 0, i.e. raw coordinates). In our experiments, we use raw coordinates, as preliminary tests showed no benefit from Fourier features on our datasets.

Loss and evaluation We train GNOT with the same loss as our main model on each benchmark—L_1 throughout, except the L_2 used on CO_2—predicting a per-step delta in z-score space. Evaluation uses a 60-step autoregressive rollout with vRMSE as the primary metric, identical to all other baselines.

Parameter budget We size the GNOT architecture to approximately match the parameter count of our main model on each benchmark:

- Acoustic scattering (256×256): $d = 256$, $N_{\text{blocks}} = 5$, 1 MLP hidden layer $\approx$ 18.8M parameters (target: $<$20M).
- FlowBench FPO (256×1024): $d = 256$, $N_{\text{blocks}} = 6$, 2 MLP hidden layers $\approx$ 36M parameters (target: $\sim$39M).
- CO_2–water flow in porous media (512×512): $d = 256$, $N_{\text{blocks}} = 6$, 2 MLP hidden layers $\approx$ 35M parameters (target: $\sim$37.7M).

All configurations use a single attention head, no MoE, and no Fourier positional features.

Training On acoustic scattering and FlowBench, we train GNOT for 50k steps on a single GPU with a cosine learning-rate schedule (peak 1.875×10^{-4}, 100-step warmup) and gradient clipping at norm 10. Validation is performed every 500 steps with 60-step rollouts. On CO_2, GNOT instead uses the same optimization and evaluation settings as the other CO_2 models (Tab. 11): $T_{\text{in}}=4$, a constant learning rate of 5×10^{-5} after warmup, gradient clipping at norm 2.0, and validation every 1000 steps.

B.4 Experimental Settings

Compute All experiments were run on NVIDIA H100 GPUs. Number of GPUs used for training and approximate wall-clock time per run were: i) Acoustic scattering: 1 GPU, 9–10 hours; ii) FlowBench: 1 GPU, 8–10 hours; iii) CO_2: 4 GPUs, 13–20 hours (2 hours for GNOT).

Table 9: Configuration for the acoustic scattering experiments from The Well (Ohana et al., 2024) in Table 1. All runs share the settings in the upper block and differ only in the lower one; head dimension is held at 128 throughout, so width and head count move together. Normalization uses The Well's `ZScoreNormalization` with statistics computed on the training split. GNOT baselines use separate architectures and training settings, given in App. B.3.1.

Data	
Datasets	acoustic scattering: maze, inclusions, discontinuous (256×256)
Split	official The Well `train` / `valid` splits
Joint training	the three datasets concatenated, sampled in proportion to size
State fields	pressure and velocity (3 channels), z-score normalized
Conditioning channels	density and speed of sound (2 channels)
Excluded trajectories	one trajectory (index 347) in inclusions, whose simulation diverges
Architecture	
Patch size	8 ($32\times32 = 1024$ patches)
Input timesteps T_{in}	3
Head dimension	128
MLP ratio	4.0
QK-norm	enabled
Temporal RoPE base θ	10 000
Spatial RoPE base θ	10 000
Cross-attention RoPE base θ	300 (cross-attention row only; see below)
Optimization	
Optimizer	AdamW, β=$(0.9, 0.999)$, ε=10^{-8}, weight decay 0.01
Learning rate	1.875×10^{-4}, constant after warmup
Warmup	100 steps, linear from 10^{-7}
Training steps	50 000
Batch size	32
Gradient clipping	global norm 10.0
Loss	per-field L_1
Precision	fp32
Training seeds	1234, 21, 67
Evaluation	
Validation windows	200 per dataset (single-dataset runs: 200; joint: 600)
Window selection	uniformly at random from all validation windows (seed 42); identical across runs
Joint aggregation	mean over the three datasets, each contributing 200 windows

Model	d_{model}	Blocks	Heads	Conditioning module
In-context	256	10	2	–
In-context‡ (wide)	384	6	3	–
In-context‡ (deep)	256	14	2	–
AdaNorm (global)	256	10	2	MLP 256→512, 4 layers, global pool
Cross-attention	256	11	2	MLP 256→512, 4 layers, per-patch KV
`PAdaNorm` (spatial)	256	10	2	MLP 256→512, 4 layers, per-patch

Table 10: Configuration for the FlowBench FPO (Tali et al., 2025) experiments in Table 1. All runs share the settings in the upper block and differ only in the lower one; conditioning inputs are normalized with statistics computed on the training split. GNOT baselines use separate architectures and training settings, given in App. B.3.1.

Data	
Dataset	FlowBench FPO, 2D Navier–Stokes, 1024×256
Split (simulations)	881 train / 219 valid, stratified by geometry type
Trajectory length	242 steps; 236 sliding windows per trajectory
State fields	u, v, p (3 channels), z-score normalized
Conditioning channels	mask, SDF, Reynolds number (broadcast as a uniform 2D field)
Reynolds preprocessing	$\log_{10}$ then z-score normalized
SDF preprocessing	z-score normalized
Mask preprocessing	none (already binary)
Architecture	
Patch size	16 ($64\times16 = 1024$ patches)
Input timesteps T_{in}	6
Head dimension	128
MLP ratio	4.0
QK-norm	enabled
Temporal RoPE base θ	30
Spatial RoPE base θ	10 000
Cross-attention RoPE base θ	60 (cross-attention row only; see below)
Optimization	
Optimizer	AdamW, β=(0.9, 0.999), ε=10^{-8}, weight decay 0.01, fused
Learning rate	4.6875×10^{-5}, constant after warmup
Warmup	100 steps, linear from 10^{-7}
Training steps	50 000
Batch size	8
Gradient clipping	global norm 10.0
Loss	per-field L_1
Precision	fp32
Training seeds	1234, 21, 67
Evaluation	
Validation windows	60
Window selection	uniformly at random from all validation windows (seed 42); identical across runs

Model	d_{model}	Blocks	Heads	Conditioning module
In-context	384	10	3	–
In-context‡ (wide)	512	7	4	–
In-context‡ (deep)	384	13	3	–
AdaNorm (global)	384	10	3	MLP 256→512, 4 layers, global pool
Cross-attention	384	11	3	MLP 256→512, 4 layers, per-patch KV
`PAdaNorm` (spatial)	384	10	3	MLP 256→512, 4 layers, per-patch

B.5 Computational Cost of Conditioning

This section derives the forward cost of each strategy and reports measured counts for the exact configurations that were trained (Tab. 12).

Counting convention For each architecture in the paper, we count forward-pass FLOPs with PyTorch's `FlopCounterMode`. These only account for matrix multiplications, convolutions, and attention matrix products; the element-wise operations we do not count (normalization, activations, and the modulation itself) amount to less than 1% of the total in our measurements.

We write $N = N_x N_y$ for the number of patches per frame, T_{in} for the history length, D for the model width, $D_{\text{ff}} = rD$ for the feed-forward width ($r = 4$), N_{blocks} for the number of blocks, D_h for the width of the conditioning encoder, L_c for the number of its hidden layers, D_c for the embedding dimension of a conditioning token, and $C_{\text{state}}, C_{\text{cond}}$ for the number of state and conditioning

Table 11: Configuration for the CO_2–water flow in porous media (Abdellatif et al., 2026) experiments in Table 1. All runs share the settings in the upper block and differ only in the lower one; conditioning inputs are normalized with statistics computed on the training split. GNOT baselines use separate architectures and training settings, given in App. B.3.1.

Data	
Dataset	CO_2–water flow in porous media, 512×512
Split (simulations)	500 train / 124 valid, split by base geometry
Split procedure	100 base geometries permuted with seed 42; 80 train / 20 valid
Trajectory length	100 steps; 96 sliding windows per trajectory
State fields	$\alpha_{\text{water}}, p, u_x, u_y$ (4 channels), z-score normalized
Conditioning channels	binary pore mask (1 = pore, 0 = rock) and normalized x, y coordinates in $[-1, 1]$ (3 channels)
Mask preprocessing	none (already binary)
Architecture	
Patch size	8 ($64\times64 = 4096$ patches)
Input timesteps T_{in}	4
Head dimension	128
MLP ratio	4.0
QK-norm	enabled
Temporal RoPE base θ	9
Spatial RoPE base θ	60
Cross-attention RoPE base θ	60 (cross-attention row only)
Optimization	
Optimizer	AdamW, β=(0.9, 0.999), ε=10^{-8}, weight decay 0.01, fused
Learning rate	5×10^{-5}, constant after warmup
Warmup	100 steps, linear from 10^{-7}
Training steps	50 000
Batch size	16 (4 GPUs $\times$ 4)
Gradient clipping	global norm 2.0
Loss	L_2 over all pixels (not masked to the pore space)
Precision	fp32
Training seeds	1234, 21, 67
Evaluation	
Validation windows	60
Window selection	uniformly at random from all validation windows (seed 42); identical across runs
Rollout horizon	60 steps, predicted deltas *not* masked to the pore space
Validation interval	1000 steps (checkpointing tied to the same grid)
Checkpoint selection	best unmasked $T\in[21{:}60]$ vRMSE over the 50 checkpoints, per seed

Model	d_{model}	Blocks	Heads	Conditioning module
In-context	384	10	3	–
In-context‡ (wide)	512	7	4	–
In-context‡ (deep)	384	13	3	–
AdaNorm (global)	384	10	3	MLP 256→512, 4 layers, global pool
Cross-attention	384	10	3	MLP 256→512, 4 layers, per-patch KV
`PAdaNorm` (spatial)	384	10	3	MLP 256→512, 4 layers, per-patch

channels. A linear map $\mathbb{R}^a \to \mathbb{R}^b$ applied to M tokens costs $2Mab$, and attention between M queries and M' keys of total width D costs $4MM'D$, counting $QK^\top$ and the value matrix product.

Backbone Here, we derive the FLOPs for the forward pass of the model backbone, i.e., ignoring the conditioning mechanisms. Temporal attention reshapes the tokens to (N, T_{in}, D) and applies four $D \to D$ projections followed by attention over T_{in} positions, which costs $8T_{\text{in}}ND^2 + 4NT_{\text{in}}^2D$ per block. The spatial sublayer of Eq. 9 uses a single fused projection $D \to 3D + 2D_{\text{ff}}$ shared by the attention and feed-forward branches, two output projections, and attention over N positions, costing $T_{\text{in}}ND(8D + 6D_{\text{ff}}) + 4T_{\text{in}}N^2D$. With $r = 4$ the two sublayers together give

$$\Phi_{\text{block}} = \underbrace{40\,T_{\text{in}}ND^2}_{\text{projections}} + \underbrace{4\,T_{\text{in}}N^2D}_{\text{spatial attention}} + \underbrace{4\,NT_{\text{in}}^2D}_{\text{temporal attention}}, \tag{15}$$

Outside the transformer blocks, the tokenizer costs $2T_{\text{in}}NDCP^2$ with $C = C_{\text{state}} + C_{\text{cond}}$ for In-context and $C = C_{\text{state}}$ otherwise, and the decoder, applied to the last frame only, costs $2ND\frac{D}{4}s_1^2 + 2Ns_1^2\frac{D}{4}C_{\text{state}}s_2^2$ for the two transposed convolutions of stride $s_1 s_2 = P$. Every model of Tab. 1 except GNOT shares this architecture; they differ only in the conditioning term below. GNOT's FLOPs are measured directly (Tab. 12).

Global AdaNorm Average pooling reduces the conditioning fields to a single vector before the encoder, so the whole conditioning path acts on one token and one shared modulation network:

$$\Phi_{\text{AdaNorm}} = 2\big(C_{\text{cond}}D_h + (L_c-1)D_h^2 + D_hD_c\big) + 2\big(D_cD_h + (L_c-1)D_h^2 + 3D_hD\big), \quad (16)$$

which is $O(1)$ in both N and T_{in}, and hence only 0.004 GFLOPs.

`PAdaNorm` The shared encoder f_θ maps each of the N patches from $C_{\text{cond}}P^2$ raw pixels to D_h, and each block applies its own linear projection $g_\phi^{(\ell)} : D_h \to 3D$ per patch:

$$\Phi_{\texttt{PAdaNorm}} = \underbrace{2N\big(C_{\text{cond}}P^2D_h + 2D_cD_h + 2(L_c-1)D_h^2\big)}_{f_\theta\text{, once per forward}} + \underbrace{6\,N_{\text{blocks}}NDD_h}_{g_\phi^{(\ell)}\text{, one per block}}. \quad (17)$$

The resulting cost is *linear* in the number of patches, *independent of the history length* because one set of modulation parameters is broadcast over all T_{in} timesteps, and is also independent of the rollout length when the conditioning is static.

Cross-attention The conditioning fields are tokenized on the same patch grid by a dedicated conditioning encoder, and each block allows the $T_{\text{in}}N$ state tokens to attend to the conditioning tokens via query, key, value, and output projections. Hence

$$\Phi_{\text{cross}} = 2N\big(C_{\text{cond}}P^2D_h + (L_c-1)D_h^2 + D_hD_c\big) + 4\,N_{\text{blocks}}T_{\text{in}}ND\,(D + D_c + N), \quad (18)$$

where the three terms in the bracket are the query and output projections, the key and value projections, and the attention products. Unlike Eq. 17, the computation cost of cross-attention is linear in T_{in} and quadratic in N, and in cases where conditioning is static, only the key and value projections are independent of the rollout length.

Scaling Ignoring the encoder, the ratio of the per-block conditioning costs of cross-attention Eq. 18 and `PAdaNorm` Eq. 17 is significantly in favor of `PAdaNorm`

$$\frac{4T_{\text{in}}ND(D + D_c + N)}{6ND_hD} = \frac{2T_{\text{in}}(D + D_c + N)}{3D_h}, \quad (19)$$

equal to 6.0 on the acoustic configuration, 13.0 on FlowBench, where the history is twice as long, and 24.7 on CO_2, which has four times as many patches per frame and four input timesteps.

Amortization over a rollout The conditioning fields are static for the acoustic scattering, FlowBench, and CO_2 datasets, allowing some of the conditioning computations to be reused across the rollout. For `PAdaNorm`, the entire conditioning computation can be cached once and reused: the computation cost of an S-step rollout is $S\,\Phi_{\text{backbone}} + \Phi_{\texttt{PAdaNorm}}$, so its marginal per-step cost in counted FLOPs equals that of an unconditioned model of the same size; the only remaining per-step overhead is the element-wise modulation, which `FlopCounterMode` does not count. For cross-attention, only the key and value projections can be reused across rollout steps; queries, attention products, and the output projection depend on the current state and are done S times. The gap, therefore, grows linearly with the rollout length: cross-attention repeats its state-dependent conditioning cost at every step, whereas `PAdaNorm` pays its conditioning cost once (Tab. 12).

Table 12: Forward-pass and rollout cost of each conditioning strategy, for the models reported in Tab. 1. Batch size 1, one multiply–add counted as two FLOPs. *cond. (static)* is the part of the conditioning cost that does not depend on the state and is computed once per rollout; *cond. (state-dep.)* is the part recomputed at every step, such as the cross-attention queries. The rollout columns are accordingly $\Phi_{\text{step}} = \Phi_{\text{fwd}} - \Phi_{\text{static}}$ and $60\,\Phi_{\text{step}} + \Phi_{\text{static}}$. App. B.5 gives the counting convention and the derivations.

Conditioning	#Params	Forward pass (GFLOPs)			60-step rollout	
		total	cond. (static)	cond. (state-dep.)	GFLOPs per step	TFLOPs
Acoustic scattering — 256×256, P=8, T_{in}=3						
In-context	13.3 M	113.5	–	–	113.5	6.81
In-context (wide)	18.0 M	138.9	–	–	138.9	8.33
In-context (deep)	18.6 M	158.6	–	–	158.6	9.52
AdaNorm (global)	15.5 M	113.3	< 0.01	–	113.3	6.80
Cross-attention	18.5 M	179.7	10.9	44.3	168.9	10.14
GNOT	18.8 M	41.2	2.3	2.0	38.8	2.33
`PAdaNorm` (spatial-only, ours)	19.1 M	125.2	11.9	–	113.3	6.81
FlowBench FPO — 1024×256, P=16, T_{in}=6						
In-context	30.8 M	468.2	–	–	468.2	28.09
In-context (wide)	38.6 M	553.7	–	–	553.7	33.22
In-context (deep)	39.6 M	606.1	–	–	606.1	36.36
AdaNorm (global)	32.9 M	464.6	< 0.01	–	464.6	27.87
Cross-attention	40.2 M	686.0	29.3	146.2	656.7	39.43
GNOT	36.1 M	77.1	3.1	2.4	74.1	4.45
`PAdaNorm` (spatial-only, ours)	38.6 M	481.2	16.6	–	464.6	27.89
CO_2–water flow in porous media — 512×512, P=8, T_{in}=4						
In-context	29.9 M	2005.2	–	–	2005.2	120.31
In-context (wide)	37.3 M	2175.5	–	–	2175.5	130.53
In-context (deep)	38.8 M	2604.7	–	–	2604.7	156.28
AdaNorm (global)	32.3 M	2002.8	< 0.01	–	2002.8	120.17
Cross-attention	35.8 M	3203.0	72.7	1127.4	3130.2	187.89
GNOT	35.3 M	301.2	11.1	9.7	290.0	17.41
`PAdaNorm` (spatial-only, ours)	37.7 M	2067.0	64.2	–	2002.8	120.23

C Ablation details

C.1 Global AdaNorm Baseline

As one of the baselines, we consider a globally conditioned adaptive-normalization strategy in which the spatially varying field $\mathbf{C} \in \mathbb{R}^{C_{\text{cond}} \times H \times W}$ is first reduced to a single global conditioning vector via spatial average pooling:

$$\mathbf{c} = \frac{1}{HW} \sum_{p,q} \mathbf{C}_{:,p,q} \in \mathbb{R}^{C_{\text{cond}}}, \tag{20}$$

followed by a shallow MLP f_θ that maps $\mathbf{c}$ to a latent representation $\mathbf{c}' = f_\theta(\mathbf{c}) \in \mathbb{R}^{D_c}$. Unlike `PAdaNorm`, this conditioning vector is spatially uniform: all tokens within a given sample receive identical modulation parameters, discarding the spatial structure of the conditioning fields.

A single MLP g_ϕ, shared across all blocks, then maps $\mathbf{c}'$ to scale, shift and gate parameters,

$$\boldsymbol{\gamma},\ \boldsymbol{\beta},\ \boldsymbol{\alpha} = g_\phi(\mathbf{c}'), \tag{21}$$

which modulate both the temporal and the spatial attention sublayers of every block in the AdaLN-Zero fashion of Sec. 3.2. The modulation is therefore identical across tokens, blocks, and attention axes.

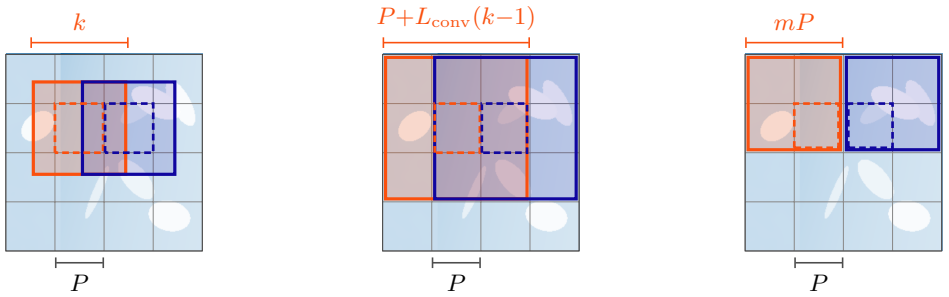
C.2 Different approaches to patching

Sliding-window MLP We keep a unique conditioning patch for each state-variable patch, with the conditioning patch overlapping neighboring patches to include more information. Let $\mathbf{C} \in$

$\mathbb{R}^{C_{\text{cond}} \times H \times W}$ denote the stacked conditioning fields, P the patch size, and $k > P$ the kernel size. For each patch index (i, j) on the grid of size $N_x \times N_y$, we extract a sliding window of size $k \times k$, centered on the corresponding patch and zero-padded at the boundaries. A shared MLP then maps each flattened window to the conditioning space: $\mathbf{c}_{ij} = \text{MLP}(\mathbf{w}_{ij}) \in \mathbb{R}^{D_h}$. The receptive field is exactly $k \times k$ pixels per token. Because the MLP operates on the full flattened window, it can learn position-specific patterns within each receptive field, at the cost of a first-layer parameter count that grows as $\mathcal{O}(C_{\text{cond}}\, k^2\, D_h)$.

Convolutional encoder We stack L_{conv} convolutional layers (kernel size k, same-padding, SiLU activations) at full spatial resolution, followed by a single strided convolution (kernel size P, stride P) that downsamples to the patch grid. The effective receptive field is $P + L_{\text{conv}}(k-1)$ pixels per token and grows linearly with depth, while weight sharing keeps the per-layer cost at $\mathcal{O}(D_h\, k^2)$, independent of the receptive field size. The trade-off is translation equivariance: unlike the MLP variant, the convolutional encoder cannot learn position-specific patterns within each patch.

Coarse conditioning patches While the former two approaches produce a unique conditioning vector per state patch, we also consider reducing the conditioning resolution relative to the state grid. Given an integer multiplier $m \geq 1$, we partition the conditioning fields into non-overlapping patches of size $mP \times mP$, each covering $m \times m$ state patches. A shared MLP maps each coarse patch to a conditioning vector $\mathbf{c}_s \in \mathbb{R}^{D_h}$, which is then broadcast identically to all m^2 state tokens it covers. Note that $m = 1$ recovers the default one-to-one alignment. Unlike the two overlapping variants, the coarse patches are not centered on the state patch they condition on, and we require m to divide both N_x and N_y.



(a) Sliding-window MLP (b) Convolutional encoder (c) Coarse conditioning

Figure 5: **Conditioning patching strategies.** Three approaches for conditioning patch design. Orange and blue regions show the input area contributing to two adjacent conditioning tokens, dashed squares mark the corresponding state patches, and the grid shows the $P \times P$ state patches. The width of the orange region is k for the sliding window, $P + L_{\text{conv}}(k-1)$ for the convolutional encoder, and mP for coarse conditioning.