PRESERVING LOCAL DETAIL IN NEURAL PDE SIMULATORS WITH STRUCTURED LATENT STATES

Anonymous authors
Paper under double-blind review

ABSTRACT

High-fidelity numerical simulation of complex physical systems often incurs prohibitive computational costs, particularly for time-dependent problems that require repeated fine-resolution updates. Neural PDE simulators provide a data-driven alternative, while latent simulators further reduce rollout cost by evolving a spatially compressed representation of the physical field. However, strong spatial compression can remove the localized structures that require fine resolution in the first place. In this paper, we propose a neural PDE simulation approach that preserves localized fine-scale structure under strong spatial compression. Motivated by classical numerical methods, the approach emphasizes expressive local representations through two design principles: (i) a spatially structured latent state that preserves where information occurs in the physical domain, and (ii) a partition-of-unity blending scheme that reconstructs a continuous physical field from local latent representations. We instantiate this approach in **S**patial **PA**tch **N**eural **S**imulator (**SPANS**), an efficient latent simulator for high-resolution, time-dependent PDEs. Its architecture couples short- and long-range attention processor with an expressive decoder whose patch-conditioned weights vary with spatial position. Across seven time-dependent simulation problems spanning structured grids, irregular meshes, and point clouds, SPANS achieves strong rollout accuracy, preserving localized spatial detail even under substantial compression, while also maintaining lower rollout time than most methods.

1 INTRODUCTION

High-fidelity simulation of time-dependent physical systems is central to many areas of science and engineering, including fluid dynamics, wave propagation, heat transfer, and solid mechanics (Evans, 2010; LeVeque, 2007; Hughes, 2000). These systems are commonly modeled through Partial Differential Equations (PDE) and solved by discretizing the physical domain in space and time (LeVeque, 2007; Hughes, 2000). Accurate simulation often requires fine spatial discretizations when the dynamics contain localized structures, such as thin transported filaments and wakes in fluids, wrinkles and creases in forming (Boisse et al., 2021), or deformation concentrated around contact (Yu et al., 2024). The resulting high-resolution spatial domains can make repeated simulation expensive, particularly over long time horizons.

Neural simulators provide a data-driven alternative and have been developed across regular grids, meshes, and irregular spatial samples. One major direction is neural operator learning, where methods such as FNO (Li et al., 2021) and CNO (Raonić et al., 2023) learn direct mappings between spatial fields on structured grids; while later approaches such as GINO (Li et al., 2023) and RIGNO (Mousavi et al., 2025) extend operator learning to more general geometries. In parallel, MGN (Pfaff et al., 2021) showed that graph neural networks can model time-dependent physical dynamics directly on arbitrary mesh discretizations. More recent attention-based models such as Transolver (Wu et al., 2024) and GAOT (Wen et al., 2025) further extend neural simulation to irregular spatial samples and complex geometries. However, for autoregressive time-dependent simulation, error accumulation over rollouts remains an important challenge (Brandstetter et al., 2022; Hoang et al., 2026). Moreover, many of these models need to evolve or repeatedly reconstruct the complete spatial field, so their rollout cost remains tied to the resolution of the physical discretization.

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.

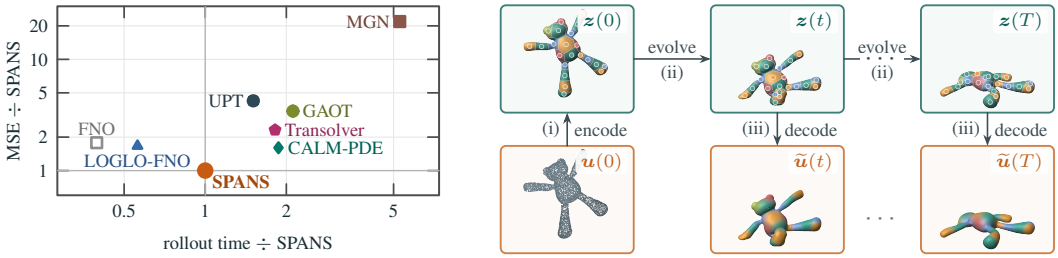


Figure 1: **Overview of SPANS.** ***Left:*** SPANS achieves the lowest aggregate rollout error across seven diverse time-dependent PDE problems while maintaining lower rollout time than all methods except FNO and LOGLO-FNO (which are faster but less accurate and are limited to regular-grid inputs). ***Right:*** The architecture consists of three main blocks: (i) the *encoder* first maps the observed field to a set of spatial patch states. (ii) The *processor* evolves this coarse latent representation over time, from which (iii) the *decoder* reconstructs the continuous physical field on the original domain.

A natural way to reduce this cost is to evolve the dynamics in a compressed latent space. Recent approaches represent the physical field through compact global states, latent tokens, or spatially learned latent points (Wu et al., 2022; Alkin et al., 2024; Wang & Wang, 2024; Serrano et al., 2024; Hagnberger et al., 2025). While these approaches can substantially reduce rollout cost, they can struggle with fine-scale features, as the compressed representation must preserve both their local structure and location in the physical domain. We focus on preserving such localized, time-dependent fine-scale structure under strong spatial compression.

To address this challenge, we introduce Spatial Patch Neural Simulator (SPANS), illustrated in Fig. 1. Rather than compressing the physical field into abstract latent tokens or learned points, SPANS encodes it as a set of spatial patch states that preserve a coarse spatial organization of the original field. Each patch state represents a local region, and a processor evolves the collection of patch states through time using a short-range and long-range attention mechanism. To reconstruct the physical field, neighboring patch states are smoothly combined using partition-of-unity (POU) weights, following the classical principle of assembling a global approximation from local ones (Babuška & Melenk, 1997). A neural field then maps the resulting local representations to solution values at arbitrary query positions. This design enables expressive local representations to recover fine-scale spatial structure while keeping the latent spatial representation coarse.

Because the physical field is reconstructed only from the evolved patch states, the latent rollout remains entirely in this coarse representation. This decouples the resolution of the latent dynamics from that of the reconstructed field and enables training on multistep sequences using only a subset of patch locations. At inference time, the same model can evaluate the solution at arbitrary queried coordinates and resolution. We evaluate SPANS across structured grids, irregular meshes, and point clouds and show that it preserves time-dependent localized fine-scale structure under strong spatial compression while remaining computationally efficient as the observation resolution increases.

An in-depth discussion of related work is provided in Appendix A.

Our contributions: **(i)** we formulate a general framework for learning dynamics in a structured latent space, motivated by Galerkin projection and the partition-of-unity method, that decouples the resolution of the latent dynamics from that of the reconstructed physical field; **(ii)** we instantiate this framework in SPANS, which encodes observations into spatial patch states, evolves them using a novel short- and long-range attention mechanism, and reconstructs a continuous physical field through a partition-of-unity neural-field decoder; and **(iii)** we conduct a comprehensive empirical evaluation across seven time-dependent problems on grids, meshes, and point clouds, demonstrating accurate rollouts, efficient computation and preservation of localized fine-scale structure under substantial spatial compression, together with detailed ablations supporting our main design choices.

2 Problem Statement

A time-dependent PDE describes the evolution of a physical quantity $u(\boldsymbol{x}, t)$ over a spatial domain $\boldsymbol{x} \in \Omega \subset \mathbb{R}^d$ and a time interval $t \in [0, T]$ with predefined initial and boundary conditions. We

consider the general form:

$$\partial_t u = F\big(u, \nabla u, \nabla^2 u, \dots\big), \qquad (\boldsymbol{x}, t) \in \Omega \times (0, T] \tag{1}$$

where $u : \Omega \times [0, T] \to \mathbb{R}$ is the unknown solution we aim to approximate (for simplicity, and without loss of generality, we focus our discussion on scalar fields) and F is a possibly nonlinear operator that depends on u and its spatial derivatives.

To solve Eq. (1) for a given F, one typically resorts to numerical methods. One of the most common approximation approaches is the method of lines, which involves discretizing the spatial domain into a finite number of points and then solving the resulting system of ordinary differential equations (ODEs) over time, leading to a system of the form:

$$\dot{\boldsymbol{u}}_N(t) = \boldsymbol{f}\big(\boldsymbol{u}_N(t)\big), \qquad \boldsymbol{u}_N(t) = \big[u(\boldsymbol{x}_1, t), \dots, u(\boldsymbol{x}_N, t)\big]^\top, \tag{2}$$

where N is the number of discretization points, $\boldsymbol{u}_N(t) \in \mathbb{R}^N$ is the spatial state vector, and $\boldsymbol{f}$ represents the spatial discretization of F.

Assume operator F in Eq. (1) is unknown. Given a dataset of K trajectories of the spatial state vector, $\{\boldsymbol{u}_N^{(k)}(t)\}_{k=1}^K$, where $\boldsymbol{u}_N^{(k)}(t)$ denotes the observation of $\boldsymbol{u}_N(t)$ for the k-th trajectory at time t, our goal is to learn a surrogate model, $\boldsymbol{g}_\theta$, that can efficiently approximate $\boldsymbol{u}_N$, given the initial condition, $\boldsymbol{u}_N(0)$, by minimizing the following loss function:

$$\mathcal{L}_{\text{multistep}}(\theta) = \frac{1}{K}\sum_{k=1}^{K}\frac{1}{T^{(k)}}\sum_{t=1}^{T^{(k)}}\left\|\boldsymbol{u}_N^{(k)}(t) - \boldsymbol{g}_\theta\left(t; \boldsymbol{u}_N^{(k)}(0)\right)\right\|_2^2, \tag{3}$$

where $T^{(k)}$ is the length of the k-th trajectory and θ are the parameters of the solution map.

3 Learning Dynamics on a Structured Latent Space

Learning the solution map $\boldsymbol{g}_\theta$ is often computationally prohibitive when N is large. In this section, we propose an alternative approach and show that by employing a structured coarse representation of the solution, we can instead learn the dynamics on a latent space more efficiently and yet expressively enough to capture the fine-scale dynamics of the solution.

Projecting to a linear system of ODEs. The method of lines yields a finite-dimensional dynamical system for a spatially discretized approximation of the PDE solution. A common way to construct this spatial discretization is through Galerkin projection (Brenner & Scott, 2008), which represents the approximate solution using time-dependent coefficients of spatial basis functions. Let H denote the infinite-dimensional function space in which the PDE solution evolves, and let $V_M = \operatorname{span}\{\psi_1, \dots, \psi_M\} \subset H$ be a finite-dimensional approximation space. Galerkin projection seeks an approximation $u_M(\cdot, t) \in V_M$ of the form:

$$u_M(\boldsymbol{x}, t) = \sum_{m=1}^{M} z_m(t)\psi_m(\boldsymbol{x}), \quad \text{such that} \quad u(\boldsymbol{x}, t) \approx u_M(\boldsymbol{x}, t). \tag{4}$$

Here, the basis functions ψ_j encode the spatial dependence, while the coefficients $z_j(t)$ evolve in time. Thus, $u_M(\cdot, t)$ remains a function over the physical domain, but its temporal evolution is completely determined by the finite-dimensional coefficient vector $\boldsymbol{z}(t) = [z_1(t), \dots, z_M(t)]^\top \in \mathbb{R}^M$, which we interpret as the latent state.

Projecting the PDE in Eq. (1) onto V_M determines the evolution of these coefficients and yields a finite-dimensional system of ordinary differential equations:

$$\boldsymbol{\Phi}\dot{\boldsymbol{z}}(t) = \boldsymbol{r}\big(\boldsymbol{z}(t)\big), \quad \text{or, assuming that } \boldsymbol{\Phi} \text{ is invertible,} \quad \dot{\boldsymbol{z}}(t) = \boldsymbol{\Phi}^{-1}\boldsymbol{r}\big(\boldsymbol{z}(t)\big), \tag{5}$$

where $\boldsymbol{\Phi} \in \mathbb{R}^{M\times M}$ is the mass matrix and $\boldsymbol{r} : \mathbb{R}^M \to \mathbb{R}^M$ is the projected residual. Consequently, $t \mapsto \boldsymbol{z}(t)$ defines a trajectory in the finite-dimensional coefficient space $\mathbb{R}^M$.

The corresponding discretized field values can be recovered by evaluating the Galerkin approximation at the set of spatial locations $\{\boldsymbol{x}_i\}_{i=1}^N$:

$$\boldsymbol{u}_N(t) = \Psi\boldsymbol{z}(t), \quad \text{with } [\boldsymbol{u}_N(t)]_n = u_M(\boldsymbol{x}_n, t) \text{ and } \Psi_{nm} = \psi_m(\boldsymbol{x}_n), \tag{6}$$

and where $\Psi \in \mathbb{R}^{N\times M}$. Thus, $\boldsymbol{z}(t) \in \mathbb{R}^M$ and $\boldsymbol{u}_N(t) \in \mathbb{R}^N$ are, respectively, the coefficient-space and pointwise representations of the same approximate field $u_M(\cdot, t)$.

This classical construction motivates our neural formulation. Rather than prescribing a global basis and deriving the corresponding coefficient dynamics $\boldsymbol{\Phi}^{-1}\boldsymbol{r}$, we learn a continuous non-linear dynamic function parameterized by θ_{dyn}: $f_{\theta_{\text{dyn}}} : \mathbb{R}^M \to \mathbb{R}^M$, which tracks the evolution of the projected state:

$$\dot{\boldsymbol{z}}(t) = f_{\theta_{\text{dyn}}}\left(\boldsymbol{z}(t)\right). \tag{7}$$

Solving this latent ODE yields $\boldsymbol{z}(t)$, from which the field can be reconstructed. However, a single global projected state compresses information from the entire domain into a shared representation, requiring it to encode both the structure and spatial location of every local feature. Under strong spatial compression, such representation may fail to preserve both the local structure of fine-scale features and where they occur in the physical domain. We instead propose an alternative approach. We distribute the compression over a coarse spatial cover, where each patch state summarizes a local region. This preserves the coarse location of information, while expressive neural representations that recover fine-scale structure within each region are smoothly assembled into a global field.

Partition-of-unity. To provide this local-to-global structure, we propose a neural adaptation of the partition-of-unity method (PUM) (Melenk & Babuška, 1996; Babuška & Melenk, 1997). The classical PUM smoothly assembles local approximations defined over overlapping spatial patches into a global field. Let Ω be covered by P overlapping patches $\{\Omega_p\}_{p=1}^P$, where $P \ll N$. Each patch p is associated with a local latent approximation $q_p(\cdot, t)$, and is parameterized as

$$q_p(\boldsymbol{x}, t) = \sum_{j=1}^{J} z_{p,j}(t) b_{p,j}(\boldsymbol{x}). \tag{8}$$

where $b_{p,j}$ is the j-th local basis function of the p-th patch, and $z_{p,j}$ is its coefficient. Then, we obtain the global field, denoted $y(\boldsymbol{x}, t)$, by blending the local approximations:

$$y(\boldsymbol{x}, t) = \sum_{p=1}^{P} w_p(\boldsymbol{x}) q_p(\boldsymbol{x}, t) = \sum_{p=1}^{P} w_p(\boldsymbol{x}) \sum_{j=1}^{J} z_{p,j}(t) b_{p,j}(\boldsymbol{x}), \tag{9}$$

where $\{w_p\}_{p=1}^P$ are convex combination weights, i.e., $w_p(\boldsymbol{x}) \geq 0$ and $\sum_{p=1}^P w_p(\boldsymbol{x}) = 1$.

In contrast to Eq. (7), where the approximation quality is solely controlled by the number of basis functions M, Eq. (9) introduces two complementary parameters: the number of patches P, which controls the spatial resolution of the latent field; and the dimension of the local latent state J, which controls the expressivity of each local representation. Collecting the local coefficient vectors gives: $\boldsymbol{Z}(t) = [\boldsymbol{z}_1(t), \ldots, \boldsymbol{z}_P(t)]^\top \in \mathbb{R}^{P\times J}$. When the local basis functions are explicit and $M = PJ$, the latent $\boldsymbol{Z}(t)$ can be seen as a blockwise organization of the global Galerkin coefficient vector, i.e., $\boldsymbol{z}(t) = \text{vec}\big(\boldsymbol{Z}(t)\big) \in \mathbb{R}^{PJ}$.

In our neural implementation, the rows of $\boldsymbol{Z}(0)$, which play the analogous role of local coefficient blocks, are learned through an encoder. Let $\mathcal{E}_{\theta_{\text{enc}}}$ denote the learned encoder. Given the initial observations $\{(\boldsymbol{x}_i, u^{(k)}(\boldsymbol{x}_i, 0))\}_{i=1}^N$, the encoder constructs a latent state per patch:

$$\boldsymbol{z}_p(0) = \mathcal{E}_{\theta_{\text{enc}}}\left(\boldsymbol{x}_i, u\left(\boldsymbol{x}_i, 0\right)\right), \quad p = 1, \ldots, P. \tag{10}$$

The latent dynamics is then learned with a neural sequence model, $f_{\theta_{\text{dyn}}}$, which allows to evolve the latent states at inference time:

$$\dot{\boldsymbol{Z}}(t) = f_{\theta_{\text{dyn}}}(\boldsymbol{Z}(t)). \tag{11}$$

Once we have a solution, $\boldsymbol{Z}(t)$, the global latent field is assembled by blending the patches with the PUM weights.

Reconstructing the physical field. In classical PUM, the reconstructed field $y(\boldsymbol{x}, t)$ is obtained by combining local basis expansions and can be used directly as an approximation of the physical field $u(\boldsymbol{x}, t)$. Its approximation quality depends on the choice and number of local basis functions.

In our neural implementation, rather than prescribing these basis functions $\{\boldsymbol{b}_p = (b_{p,j})_{j=1}^J\}_{p=1}^P$, we use learnable parametric functions, denoted $\{h_{\theta_{p,\text{dec}}}\}_{p=1}^P$. Since accurately reconstructing the

global field requires expressive local functions, which may be expensive to evaluate, we propose a decoder with two-stages:

$$y(\boldsymbol{x},t)=\sum_{p=1}^{P} w_p(\boldsymbol{x})h_{p,\theta_{\mathrm{dec}}}\big(\boldsymbol{x},\boldsymbol{z}_p(t)\big),\qquad \widetilde{u}(\boldsymbol{x},t)=h_{\theta_{\mathrm{rec}}}\big(\boldsymbol{x},y(\boldsymbol{x},t)\big). \tag{12}$$

Here, we keep $h_{p,\theta_{\mathrm{dec}}}$ lightweight, and assemble their outputs with the PoU weights into an intermediate global field $y(\boldsymbol{x},t)$. The expressive part of the decoder is instead delegated to $h_{\theta_{\mathrm{rec}}}$, which maps the intermediate field $y(\boldsymbol{x},t)$ to the physical field. Although this transformation may be computationally expensive, it is applied only after assembly and therefore evaluated once per query location. The complete set of decoder parameters is therefore $\theta_{\mathrm{dec}}=\{\,\{w_p\}_{p=1}^{P},\{\theta_{p,\mathrm{dec}}\}_{p=1}^{P},\theta_{\mathrm{rec}}\}$.

In summary, Galerkin projection motivates representing the evolution of a spatial field through a finite-dimensional coefficient state. The partition-of-unity method then motivates organizing this state into spatially localized blocks, each associated with a local approximation, and assembling these local representations into a global field through normalized overlapping weights. Finally, instead of specifying the basis functions explicitly for Galerkin or partition-of-unity, our approach learns both the encoder, latent dynamics and local reconstruction directly from data. The parameters of the solution map, $\boldsymbol{g}_\theta$, is the combination of the parameters of the encoder, the dynamics model, and the decoder: $\theta=\{\theta_{\mathrm{enc}},\theta_{\mathrm{dyn}},\theta_{\mathrm{dec}}\}$.

4 Theoretical Analysis

We now establish an approximation bound for our approach from Section 3. We consider a scalar field; the same argument holds for vector-valued fields. We start with notation and assumptions.

Let $\boldsymbol{z}^\star(t)$ be the Galerkin trajectory of Eq. (5) for the PUM basis $\{w_p b_{p,j}\}$, i.e., $\dot{\boldsymbol{z}}^\star=f(\boldsymbol{z}^\star)$ with $f(\boldsymbol{z}^\star):=\boldsymbol{\Phi}^{-1}\boldsymbol{r}$, and let $y^\star$ be its field in Eq. (9). The model $f_{\theta_{\mathrm{dyn}}}$ instead produces the trajectory $\boldsymbol{z}(t)$ and reconstruction $\widetilde{u}$ via Eqs. (10) to (12). Define the norm: $\|v\|_{[0,T]}:=\sup_t\|v(\cdot,t)\|_{L^2(\Omega)}$.

Assumption 1 (Galerkin stability)**.** *The Galerkin approximation of the underlying PDE with solution u is stable:* $\exists B_T\geq 0\ :\ \|u-y^\star\|_{[0,T]}\leq B_T\inf_{y\in C([0,T];V_{\mathrm{PU}})}\|u-y\|_{[0,T]}$.

Assumption 2 (Latent-dynamics stability)**.** *There exists a compact set $\mathcal{K}\subset\mathbb{R}^{PJ}$ such that $\boldsymbol{z}^\star(t)$, $\boldsymbol{z}(t)\in\mathcal{K}$ for all $t\in[0,T]$. Moreover, $f_{\theta_{\mathrm{dyn}}}$ is L_{dyn}-Lipschitz on $\mathcal{K}$, meaning that*

$$\left\|f_{\theta_{\mathrm{dyn}}}(\boldsymbol{\zeta}_1)-f_{\theta_{\mathrm{dyn}}}(\boldsymbol{\zeta}_2)\right\|_2\leq L_{\mathrm{dyn}}\left\|\boldsymbol{\zeta}_1-\boldsymbol{\zeta}_2\right\|_2,\quad\forall\boldsymbol{\zeta}_1,\boldsymbol{\zeta}_2\in\mathcal{K}. \tag{13}$$

Finally, we define the encoding and dynamics approximation errors by $\varepsilon_{\mathrm{enc}}:=\|\boldsymbol{z}^\star(0)-\boldsymbol{z}(0)\|_2$, $\varepsilon_{\mathrm{dyn}}:=\sup_{t\in[0,T]}\left\|f\big(\boldsymbol{z}^\star(t)\big)-f_{\theta_{\mathrm{dyn}}}\big(\boldsymbol{z}^\star(t)\big)\right\|_2$.

Assumption 3 (Decoder smoothness & expressivity)**.** *$h_{\theta_{\mathrm{rec}}}$ is uniformly L_{rec}-Lipschitz in its second argument: $|h_{\theta_{\mathrm{rec}}}(\boldsymbol{x},y_a)-h_{\theta_{\mathrm{rec}}}(\boldsymbol{x},y_b)|\leq L_{\mathrm{rec}}|y_a-y_b|$, $\forall y_a,y_b\in V_{\mathrm{PU}}$. Also, for every $\varepsilon_{\mathrm{dec}}>0$, there exists a decoder such that its reconstruction error along the reference field satisfies: $\|y^\star-\widetilde{u}^\star\|_{[0,T]}\leq\varepsilon_{\mathrm{dec}}$.*

We are ready to state our result showing that the approximation error is bounded by three terms: the Galerkin–PUM approximation error, the reconstruction error of the decoder, and the error induced by the learned latent dynamics. The proof is in Appendix B.

Theorem 1 (Approximation Error)**.** *Let $u\in C([0,T];L^2(\Omega))$ be a solution to Eq.* (1) *on a bounded domain $\Omega\subset\mathbb{R}^d$. Suppose that Assumptions 1 to 3 hold and that $\varepsilon_{\mathrm{enc}}$ and $\varepsilon_{\mathrm{dyn}}$ are finite. Then:*

$$\|u-\widetilde{u}\|_{[0,T]}\leq B_T\varepsilon_{\mathrm{glo}}+\varepsilon_{\mathrm{dec}}+L_{\mathrm{rec}}C_b e^{L_{\mathrm{dyn}}T}\left(\varepsilon_{\mathrm{enc}}+T\varepsilon_{\mathrm{dyn}}\right) \tag{14}$$

The first two terms can be reduced through two complementary design choices. The Galerkin–PUM decreases as the local approximation spaces become more expressive, motivating the use of a sufficiently large number of spatial patches, while the reconstruction error motivates an expressive decoder that can recover the physical field from these local representations. This is supported by our empirical results, where increasing the number of patches and using the expressive decoder both improve approximation accuracy (Appendix H).

The final term reveals a different source of error: inaccuracies in the initial encoding and learned latent dynamics are propagated and accumulated over the rollout horizon. Its explicit dependence on T therefore supports multistep rollout training, which directly penalizes the accumulated trajectory error that arises from one-step training followed by autoregressive rollout.

5 Practical Implementation: Spatial Patch Neural Simulator

We now present SPANS, a concrete instantiation of the framework proposed in Section 3 for efficient simulation of time-dependent PDEs at high spatial resolution.

Encoder. Given the initial observations $\{(\boldsymbol{x}_i, u(\boldsymbol{x}_i, t_0))\}_{i=1}^N$, the encoder, $E_{\theta_{\text{enc}}}$, constructs a coarse spatial representation with P patches. Each patch is defined by a center, denoted $\boldsymbol{c}_p$, and its latent state $\boldsymbol{z}_p \in \mathbb{R}^J$. The centers can be obtained using any placement strategy, from a regular lattice to clustering methods (e.g., k-means), parameterized by θ_{cent}. Each center defines a spatial patch Ω_p. Each input point is assigned to exactly one patch. Let $\mathcal{I}_p$ denote the points assigned to center $\boldsymbol{c}_p$, as illustrated in the top row of Fig. 1. To obtain the latent states, since the number of points in $\mathcal{I}_p$ can vary across patches, we use a shared DeepSet (Zaheer et al., 2017) to aggregate them:

$$\boldsymbol{z}_p(0) = \text{MLP}_{\theta_{\text{out}}}\left(\bigoplus_{i\in\mathcal{I}_p} \text{MLP}_{\theta_{\text{in}}}([u(\boldsymbol{x}_i, t_0), \boldsymbol{x}_i - \boldsymbol{c}_p])\right), \tag{15}$$

where $\bigoplus$ denotes a pooling operator that is invariant to the order of input points, such as sum, mean, or max-pooling. The two MLPs are shared across all patches. Note that this encoding is performed only once at $t = 0$, the beginning of the predictive window, so the resulting coarse representation is fixed during each rollout. The encoder parameters are thus $\theta_{\text{enc}} = \{\theta_{\text{cent}}, \theta_{\text{in}}, \theta_{\text{out}}\}$.

Processor. The processor determines the evolution of the latent patch states $\boldsymbol{z}_p(t)$ via a neural latent vector field $f_{\theta_{\text{dyn}}}$ (Eq. (11)). Following the Galerkin analysis in Section 3, $f_{\theta_{\text{dyn}}}$ can be understood as learning a neural parameterization of $\boldsymbol{\Phi}^{-1}\boldsymbol{r}(\boldsymbol{z})$ (see Eq. (5)). An effective parameterization therefore depends on the structure of $\boldsymbol{\Phi}^{-1}$. Although $\boldsymbol{\Phi}$ is often sparse due to the blockwise organization of the bases, $\boldsymbol{\Phi}^{-1}$ is generally dense. However, it still retains locality, with its entries decaying with distance in the sparsity graph at a rate governed by the conditioning of $\boldsymbol{\Phi}$ (Hackbusch, 2015, Sec. 11.1). This motivates combining *short-range attention* over neighboring patches with *long-range attention* over all patch states, allowing the processor to capture strong local coupling while retaining unrestricted long-range communication:

$$\boldsymbol{Z}_{t+1} = \boldsymbol{Z}_t + \Delta_t \left(\text{Attn}_{\theta_{\text{short}}}(\boldsymbol{Z}_t) + \text{Attn}_{\theta_{\text{long}}}(\boldsymbol{Z}_t)\right), \tag{16}$$

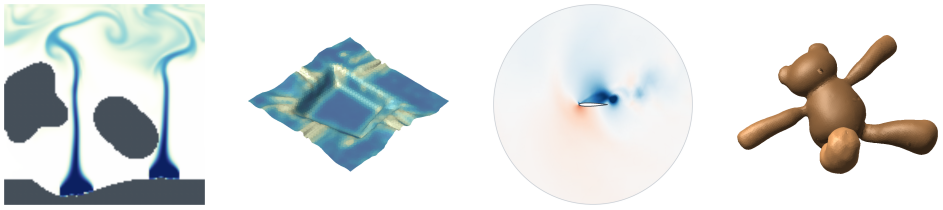
where Euler's method is used to evolve the latent states in time and Δ_t is the fixed time-step size. For second-order systems, we additionally carry a latent velocity $\boldsymbol{V}_t$ with analogous attention mechanism (see details in Appendix C.1). The latent dynamics parameters are thus $\theta_{\text{dyn}} = \{\theta_{\text{short}}, \theta_{\text{long}}\}$.

Partition-of-unity decoder. As established in Section 3, to reconstruct the fine-scale solution $u(\boldsymbol{x}, t)$, we propose a two-stages decoder: *(i)* parametric local functions combined with POU weights to obtain an intermediate global field, and *(ii)* a patch-conditioned neural field for reconstruction of the physical field.

For the POU weights, $w_p(\boldsymbol{x})$, we use (predetermined) distance-based Wendland weights (Wendland, 1995) that abstract the rest of the decoder from an explicit dependence on the location $\boldsymbol{x}$. This is a standard choice in the literature POU (Garmanjani et al., 2018), allowing the same construction to be applied to different input structures, such as regular grids, irregular meshes, and more general point clouds. In particular, let $\chi(r) = (1-r)_+^4(4r+1)$, where $(1-r)_+ = \max(1-r, 0)$. The unnormalized and normalized weights are defined as

$$\widetilde{w}_p(\boldsymbol{x}) = \chi\left(\|\boldsymbol{x} - \boldsymbol{c}_p\|_2 / \rho\right), \qquad w_p(\boldsymbol{x}) = \widetilde{w}_p(\boldsymbol{x}) \Big/ \sum_{p'=1}^{P} \widetilde{w}_{p'}(\boldsymbol{x}). \tag{17}$$

Next, we provide a concrete instantiation of the decoder. We extend the low-rank parameterization introduced in IGNS for time-varying matrices (Hoang et al., 2026), which combines fixed shared



(a) SmokeObs (b) FormNapf (c) Airfoil (d) Teddy

Figure 2: **Representative tasks.** SmokeObs transports thin buoyant filaments around trajectory-dependent obstacles, FormNapf develops localized wrinkles during composite forming, Airfoil models compressible flow on a mesh refined around the airfoil, and Teddy undergoes local deformation under contact. All seven tasks are shown in Appendix D.

bases with a small set of generated coefficients, and extend the same idea to make it both temporal and patch (spatial) dependent. In particular, we use a lightweight hypernetwork (Ha et al., 2017), which gives the parameters of an MLP with L layers with patch-dependent weights. Let $\mathcal{H}_\phi$ denote the hypernetwork, with parameter ϕ. Let $\boldsymbol{A}^{(\ell)}(\boldsymbol{x},t)$, $\boldsymbol{\beta}^{(\ell)}$ and $\boldsymbol{h}^{(\ell)}(\boldsymbol{x},t)$ denote the weights, bias and output of the ℓ-th layer of the MLP, respectively. Then, we have:

$$\mathcal{H}_\phi\big(\boldsymbol{z}_p(t),\boldsymbol{c}_p\big)=\Big\{\boldsymbol{A}_p^{(\ell)}(t)\Big\}_{p=1}^{L},\quad \boldsymbol{A}^{(\ell)}(\boldsymbol{x},t)=\sum_{p=1}^{P} w_p(\boldsymbol{x})\boldsymbol{A}_p^{(\ell)}(t), \tag{18}$$

$$\boldsymbol{h}^{(1)}(\boldsymbol{x},t)=\sigma^{(1)}\Big(\boldsymbol{y}(\boldsymbol{x},t)+\boldsymbol{\beta}^{(1)}\Big),\quad \boldsymbol{h}^{(\ell)}(\boldsymbol{x},t)=\sigma^{(\ell)}\left(\boldsymbol{A}^{(\ell)}(\boldsymbol{x},t)\boldsymbol{h}^{(\ell-1)}(\boldsymbol{x},t)+\boldsymbol{\beta}^{(\ell)}\right). \tag{19}$$

The intermediate global field is given by $\boldsymbol{y}(\boldsymbol{x},t)=\sum_{p=1}^{P} w_p\boldsymbol{A}^{(1)}(\boldsymbol{x},t)\boldsymbol{z}_p$, such that $\theta_{p,\mathrm{dec}}=\boldsymbol{A}_p^{(1)}$. The physical field is given by the output of the MLP: $\widetilde{\boldsymbol{u}}(\boldsymbol{x},t)=\boldsymbol{h}^{(L)}(\boldsymbol{x},t)$. See Appendix C.2 for a more detailed description of the implementation of the decoder.

6 Experiments

We evaluate SPANS on seven tasks with varying physics regimes and data representations. These tasks include two grid-based buoyant-driven smoke: SmokeGrid and SmokeObs; two composite-forming tasks: FormNapf (Mitsch & Kärger, 2025) and FormDome (Würth et al., 2026); two mesh-based flows with irregular meshes: Cylinder and Airfoil (Pfaff et al., 2021); and one 3D deforming object simulation: Teddy (Dahlinger et al., 2026). Dataset sizes, rollout horizons, and metrics are in Appendices D and G.

We compare against three complementary families of simulators. *(I) Spectral operators* FNO (Li et al., 2021) and LOGLO-FNO (Kalimuthu et al., 2025), which evolve the complete regular lattice on periodic domains. *(II) Point operators* GAOT (Wen et al., 2025) and Transolver (Wu et al., 2024), which repeatedly map between the observed spatial points and an internal representation. *(III) Latent rollouts* UPT (Alkin et al., 2024) and CALM-PDE (Hagnberger et al., 2025), which encode once and evolve a compressed latent state, making them the closest comparison to SPANS. For Transolver, we report the better-performing setting: its default one-step objective on the fluid tasks and multistep training on the forming tasks and Teddy. Table 4 in the Appendix provides a complete overview of the baseline design choices and their relation to SPANS.

All methods are evaluated by free rollout from the initial frames and till the task is completed. We use task-specific wall-clock training budgets on one NVIDIA A100. Main table results report means $\pm$ std over three seeds; ablation and scaling studies use a single seed to reduce excessive computational cost. Training details are reported in Appendices E and F.

6.1 Results

Rollout accuracy. Table 1 shows that SPANS achieves the lowest mean rollout error on most tasks. On the two smoke problems, SPANS consistently performs best, improving over the competitive

Table 1: **Accuracy.** Rollout-averaged MSE ($\times 10^{-2}$, $\times 10^{0}$ for the FormNapf and FormDome, and $\times 10^{-3}$ for Teddy), lower is better: density for the smoke tasks, position for the composite-forming tasks and Teddy, and flow state for Cylinder and Airfoil. Mean $\pm$ std over three seeds. For Transolver, * denotes multistep training variants. N/A denotes not applicable, as these neural operators support only regular grids. Best and second-best (average) results are highlighted in **orange** and **teal**, respectively.

		Structured lattices				Irregular meshes		Point cloud
	Method	**SmokeGrid**	**SmokeObs**	**FormNapf**	**FormDome**	**Cylinder**	**Airfoil**	**Teddy**
(I)	FNO	0.933 ± 0.066	N/A	1.020 ± 0.007	1.010 ± 0.052	N/A	N/A	N/A
	LOGLO-FNO	0.803 ± 0.021	N/A	1.030 ± 0.013	0.951 ± 0.031	N/A	N/A	N/A
(II)	GAOT	0.368 ± 0.007	0.545 ± 0.008	0.900 ± 0.019	0.941 ± 0.036	36.0 ± 4.2	35.7 ± 16.0	4.600 ± 2.190
	Transolver	1.370 ± 0.046	1.500 ± 0.042	$1.070 \pm 0.013^\star$	$1.020 \pm 0.043^\star$	1.13 ± 0.026	4.85 ± 0.54	$5.350 \pm 1.310^\star$
(III)	UPT	3.580 ± 2.600	3.450 ± 1.600	1.430 ± 0.031	1.050 ± 0.081	7.84 ± 1.1	12.3 ± 1.1	2.500 ± 0.360
	CALM-PDE	0.424 ± 0.016	0.785 ± 0.035	0.876 ± 0.015	0.951 ± 0.042	1.64 ± 0.13	5.55 ± 1.1	1.970 ± 0.055
	SPANS (ours)	0.231 ± 0.011	0.443 ± 0.012	0.799 ± 0.016	0.922 ± 0.044	2.40 ± 0.26	1.59 ± 0.095	0.633 ± 0.015

Table 2: **Local-detail retention.** Final-frame high-frequency magnitude ratio r_{HF}, crease-placement correlation ρ_z, and curvature-placement correlation ρ_{curv}, all best at 1. Groups and * (multistep Transolver) follow Table 1. N/A denotes not applicable.

		High-freq. ratio r_{HF}		Crease placement ρ_z		Curvature ρ_{curv}
	Method	SmokeGrid	SmokeObs	FormNapf	FormDome	Teddy
(I)	FNO	1.010 ± 0.016	N/A	0.421 ± 0.006	0.328 ± 0.027	N/A
	LOGLO-FNO	1.000 ± 0.006	N/A	0.406 ± 0.003	0.362 ± 0.003	N/A
(II)	GAOT	1.320 ± 0.006	1.527 ± 0.037	0.374 ± 0.006	0.328 ± 0.005	0.751 ± 0.037
	Transolver	1.200 ± 0.255	1.035 ± 0.052	$0.446 \pm 0.002^\star$	$0.300 \pm 0.021^\star$	$0.930 \pm 0.009^\star$
(III)	UPT	0.139 ± 0.099	0.083 ± 0.058	0.423 ± 0.006	0.380 ± 0.003	0.959 ± 0.003
	CALM-PDE	0.729 ± 0.040	0.496 ± 0.011	0.406 ± 0.007	0.376 ± 0.014	0.798 ± 0.008
	SPANS (ours)	0.952 ± 0.006	0.772 ± 0.046	0.448 ± 0.002	0.403 ± 0.008	0.941 ± 0.007

GAOT and CALM-PDE baselines. Meanwhile, the differences on the forming tasks are smaller, though SPANS still performs best on FormNapf and FormDome. For the irregular-mesh problems, SPANS achieves the lowest error on Airfoil but ranks third on Cylinder (Appendix H). Finally, on Teddy, SPANS achieves the best rollout accuracy by a clear margin over the other latent simulators. Among the baselines, GAOT and CALM-PDE are the most consistently competitive across structured tasks, while Transolver is particularly strong on irregular meshes but substantially weaker on structured grids and point clouds. UPT shows noticeably larger errors on several tasks, particularly on the smoke problems, which may reflect difficulty preserving spatial detail under strong compression into a small, spatially unstructured latent representation. The spectral baselines, limited to regular grids, are also less competitive on the smoke problems, consistent with the strongly localized dynamics of these tasks and the additional difficulty posed by non-periodic boundaries.

Local-detail retention. Rollout error alone does not reveal whether localized structures are preserved or merely averaged out. We therefore examine the high-frequency and local-structure metrics in Table 2. The baselines exhibit different failure modes: UPT and CALM-PDE tend to smooth out high-frequency content, whereas GAOT produces too much of it. On SmokeGrid, the spectral baselines FNO and LOGLO-FNO closely match the spectral magnitude but do not achieve comparably accurate rollouts. On FormNapf and FormDome, SPANS achieves the best crease-placement agreement. On Teddy, SPANS preserves the location of contact-induced curvature comparably to UPT and Transolver, while achieving roughly $4\times$ and $8\times$ lower position error, respectively (Table 1). Together, these results suggest that the benefit of the patch representation is not simply retaining more high-frequency content, but retaining it at the correct spatial locations.

Compute and data efficiency. Fig. 1-left shows the accuracy–cost trade-off across all tasks, measured by the geometric mean of each method's MSE and rollout time relative to SPANS, with whole-graph MGN (Pfaff et al., 2021) included for reference. Overall, SPANS achieves high rollout accuracy while remaining efficient. Compared to the whole-graph simulator MGN, SPANS is more

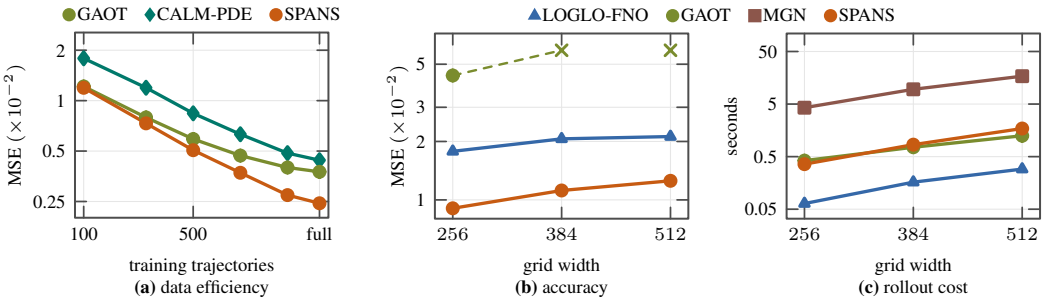


Figure 3: *(a) Data efficiency:* rollout-averaged MSE on SmokeGrid versus training-set size for SPANS and the two most accurate baselines. *(b) Resolution scaling-accuracy:* rollout-averaged MSE on SmokeGrid versus resolution, with $P = 1024$ fixed for SPANS, against the most accurate spectral and point baselines. *(c) Resolution scaling-cost:* rollout time per trajectory at these resolutions. Crosses mark settings outside the available data range, where MSE is not reported. CALM-PDE is omitted from (b,c): training exceeds device memory above 256^2.

than $5\times$ faster on average. The spectral baselines are faster through parallel spectral convolutions on regular grids, but yield substantially higher rollout errors and remain limited to regular-grid domains. We next examine whether the accuracy advantage of SPANS persists with less training data. As shown in Fig. 3(a), the advantage is already pronounced in the low-data regime: with only 100 training trajectories, SPANS reaches roughly the accuracy of CALM-PDE trained with 250 trajectories. More training data improves predictions but does not close the gap. This suggests that the structured patch representation provides a useful spatial inductive bias, allowing SPANS to learn accurate dynamics from fewer training trajectories.

Resolution scaling. Fig. 3(b) and (c) examine how the methods scale as observation resolution increases from 256^2 to 512^2, with the latent size of SPANS fixed. Across all resolutions, SPANS achieves the lowest rollout error. Its advantage over LOGLO-FNO narrows slightly at higher resolutions but remains substantial, whereas GAOT becomes unstable beyond 256^2 (Fig. 5). The rollout cost of SPANS grows only moderately: it remains comparable to GAOT, higher than LOGLO-FNO but lower than the whole-graph MGN. This increase comes primarily from decoding onto a denser query field; the processor cost remains unchanged. Consequently, SPANS can increase output resolution without enlarging its latent dynamics, although achievable accuracy eventually depends on the spatial capacity of this fixed coarse representation. We further study this capacity in Appendix H.3, where increasing the number of patches is more effective than widening patch states. Additional memory measurements, scaling results, and architectural ablations are provided in Appendix H.

7 Conclusion

We introduced the Spatial Patch Neural Simulator (SPANS), a latent neural simulator designed to preserve localized fine-scale structure under spatial compression. SPANS retains a coarse spatial organization through patch latent states, evolves them with short-range and long-range attention, and reconstructs the physical field by smoothly blending local neural representations through partition-of-unity weights. This separates coarse latent dynamics from fine spatial reconstruction and enables multistep training on a random subset of query positions. Across seven time-dependent simulation problems on grids, meshes, and point clouds, SPANS achieves strong rollout accuracy while being efficient and able to preserve localized spatial detail under substantial compression.

Limitations and future work. One limitation of SPANS is that, although the decoder is continuous in space, the latent dynamics remain tied to the fixed temporal stride of the training data, motivating future work on a continuous-time formulation that supports arbitrary temporal resolution. The encoder also assumes direct field observations; a natural extension is therefore to infer spatial patch states directly from images using the depth-estimation capabilities of modern vision encoders. Finally, although we evaluate SPANS only for forward simulation, its low-dimensional differentiable rollout is well suited to control and inverse problems, where actions, material parameters, or geometry can be optimized through the latent dynamics.

Ethics Statement

This work studies neural simulation of physical systems using existing or procedurally generated simulation data. It does not involve human participants, personal or sensitive data, or deployment in safety-critical settings. We are not aware of ethical concerns specific to the datasets, methodology, or results.

Reproducibility Statement

We describe the complete model and training objective in Section 5. Appendices C to F provide the implementation details, dataset construction and splits, hyperparameter search and selected configurations, task-specific compute budgets, and baseline implementations. We will release the code upon acceptance.

AI Use Statement

We used generative AI tools to assist with literature retrieval and review, research ideation, feedback on methodology and experiments, software implementation, and drafting and editing the manuscript for clarity. The authors verified all the cited literature. AI assisted with implementing code to convert outputs from numerical simulator into valid formats used by the neural simulators; it did not generate or alter the ground-truth data or create synthetic datasets. Ground-truth trajectories were produced by numerical simulators. All AI-assisted code was reviewed and tested. We formulated the mathematical claims and wrote the proofs ourselves. All authors reviewed the final manuscript and take responsibility for its text, claims, code, and mathematical statements.

References

Benedikt Alkin, Andreas Fürst, Simon Schmid, Lukas Gruber, Markus Holzleitner, and Johannes Brandstetter. Universal physics transformers: A framework for efficiently scaling neural operators. In *Advances in Neural Information Processing Systems*, 2024.

S. E. Arnold, M. P. F. Sutcliffe, and W. L. A. Oram. Experimental measurement of wrinkle formation during draping of non-crimp fabric. *Composites Part A: Applied Science and Manufacturing*, 82: 159–169, 2016. doi: 10.1016/j.compositesa.2015.12.011.

Ivo Babuška and Jens M. Melenk. The partition of unity method. *International Journal for Numerical Methods in Engineering*, 40(4):727–758, 1997. doi: 10.1002/(SICI)1097-0207(19970228)40:4⟨727::AID-NME86⟩3.0.CO;2-N.

Ivo Babuška, Uday Banerjee, and John E. Osborn. Survey of meshless and generalized finite element methods: A unified approach. *Acta Numerica*, 12:1–125, 2003. doi: 10.1017/S0962492902000090.

Jonghyuk Baek, Yanran Wang, and Jiun-Shyan Chen. N-adaptive Ritz method: A neural network enriched partition of unity for boundary value problems. *Computer Methods in Applied Mechanics and Engineering*, 428:117070, 2024. doi: 10.1016/j.cma.2024.117070.

Philippe Boisse, Jin Huang, and Eduardo Guzman-Maldonado. Analysis and modeling of wrinkling in composite forming. *Journal of Composites Science*, 5(3):81, 2021. doi: 10.3390/jcs5030081.

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

Susanne C. Brenner and L. Ridgway Scott. *The Mathematical Theory of Finite Element Methods*. Springer, 2008.

Philipp Dahlinger, Niklas Freymuth, Tai Hoang, Tobias Würth, Michael Volpp, Luise Kärger, and Gerhard Neumann. Context-aware learned mesh-based simulation via trajectory-level meta-learning. *Transactions on Machine Learning Research*, 2026.

Frank Doehner, Johannes Mitsch, Saksham Kiroriwal, Shahenda Youssef, Georg Zeeb, Frank Henning, Luise Kärger, and Jürgen Beyerer. Bayesian optimization and guided over-instrumented gripper design for thermoforming of composite materials. *Composites Part A: Applied Science and Manufacturing*, 207:109851, 2026. doi: 10.1016/j.compositesa.2026.109851.

Victorita Dolean, Alexander Heinlein, Siddhartha Mishra, and Ben Moseley. Multilevel domain decomposition-based architectures for physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 429:117116, 2024. doi: 10.1016/j.cma.2024.117116.

C. Armando Duarte and J. Tinsley Oden. An h-p adaptive method using clouds. *Computer Methods in Applied Mechanics and Engineering*, 139(1–4):237–262, 1996. doi: 10.1016/S0045-7825(96)01085-7.

Lawrence C. Evans. *Partial Differential Equations*, volume 19 of *Graduate Studies in Mathematics*. American Mathematical Society, 2 edition, 2010.

Tiffany Fan, Nathaniel Trask, Marta D'Elia, and Eric Darve. Probabilistic partition of unity networks for high-dimensional regression problems. *International Journal for Numerical Methods in Engineering*, 124(10):2215–2236, 2023. doi: 10.1002/nme.7207.

Gholamreza Garmanjani, Roberto Cavoretto, and Mohsen Esmaeilbeigi. A RBF partition of unity collocation method based on finite difference for initial–boundary value problems. *Computers & Mathematics with Applications*, 75(11):4066–4090, 2018. doi: 10.1016/j.camwa.2018.03.014.

David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *International Conference on Learning Representations*, 2017.

Wolfgang Hackbusch. *Hierarchical Matrices: Algorithms and Analysis*, volume 49 of *Springer Series in Computational Mathematics*. Springer, 2015. doi: 10.1007/978-3-662-47324-5.

Jan Hagnberger, Daniel Musekamp, and Mathias Niepert. CALM-PDE: Continuous and adaptive convolutions for latent space modeling of time-dependent PDEs. In *Advances in Neural Information Processing Systems*, 2025.

Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*. Springer, 2006.

Tai Hoang, Alessandro Trenta, Alessio Gravina, Niklas Freymuth, Philipp Becker, Davide Bacciu, and Gerhard Neumann. Improving long-range interactions in graph neural simulators via hamiltonian dynamics. In *International Conference on Learning Representations*, 2026.

Thomas J. R. Hughes. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Dover Publications, 2000.

Marimuthu Kalimuthu, David Holzmüller, and Mathias Niepert. LOGLO-FNO: Efficient learning of local and global features in fourier neural operators. *Transactions on Machine Learning Research*, 2025.

Elisabeth Larsson, Victor Shcherbakov, and Alfa Heryudono. A least squares radial basis function partition of unity method for solving PDEs. *SIAM Journal on Scientific Computing*, 39(6):A2538–A2563, 2017. doi: 10.1137/17M1118087.

Kookjin Lee, Nathaniel A. Trask, Ravi G. Patel, Mamikon A. Gulian, and Eric C. Cyr. Partition of unity networks: Deep hp-approximation. In *AAAI Spring Symposium on Combining Artificial Intelligence and Machine Learning with Physical Sciences*, 2021.

Randall J. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*. Society for Industrial and Applied Mathematics, 2007.

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.

Zongyi Li, Nikola Kovachki, Chris Choy, Boyi Li, Jean Kossaifi, Shourya Otta, Mohammad Amin Nabian, Maximilian Stadler, Christian Hundt, Kamyar Azizzadenesheli, and Anima Anandkumar. Geometry-informed neural operator for large-scale 3D PDEs. In *Advances in Neural Information Processing Systems*, 2023.

Miles Macklin. Warp: A high-performance python framework for GPU simulation and graphics. `https://github.com/nvidia/warp`, 2022. NVIDIA GPU Technology Conference (GTC).

J. M. Melenk and I. Babuška. Approximation with harmonic and generalized harmonic polynomials in the partition of unity method. *Computer Assisted Mechanics and Engineering Sciences*, 4(3–4): 607–632, 1997.

Jens M. Melenk and Ivo Babuška. The partition of unity finite element method: Basic theory and applications. *Computer Methods in Applied Mechanics and Engineering*, 139(1–4):289–314, 1996. doi: 10.1016/S0045-7825(96)01087-0.

Davoud Mirzaei. The direct radial basis function partition of unity (d-rbf-pu) method for solving pdes. *SIAM Journal on Scientific Computing*, 43(1):A54–A83, 2021. doi: 10.1137/19M128911X.

Johannes Mitsch and Luise Kärger. Process simulation data for thermoforming of continuous fiber-reinforced composite materials, December 2025.

Mayank Mittal, Calvin Yu, Qinxi Yu, Jingzhou Liu, Nikita Rudin, David Hoeller, Jia Lin Yuan, Ritvik Singh, Yunrong Guo, Hammad Mazhar, Ajay Mandlekar, Buck Babich, Gavriel State, Marco Hutter, and Animesh Garg. Orbit: A unified simulation framework for interactive robot learning environments. *IEEE Robotics and Automation Letters*, 8(6):3740–3747, 2023. doi: 10.1109/LRA.2023.3270034.

Ben Moseley, Andrew Markham, and Tarje Nissen-Meyer. Finite basis physics-informed neural networks (FBPINNs): A scalable domain decomposition approach for solving differential equations. *Advances in Computational Mathematics*, 49(4):62, 2023. doi: 10.1007/s10444-023-10065-9.

Sepehr Mousavi, Shizheng Wen, Levi Lingsch, Maximilian Herde, Bogdan Raonić, and Siddhartha Mishra. RIGNO: A graph-based framework for robust and accurate operator learning for PDEs on arbitrary domains. In *Advances in Neural Information Processing Systems*, 2025.

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

Banafsheh Raeisi, Mojtaba Fardi, and Mohammadreza Ahmadi Darani. RBF-based partition of unity methods for two-dimensional time-dependent PDEs: Numerical and theoretical aspects. *Mathematics and Computers in Simulation*, 226:152–171, 2024. doi: 10.1016/j.matcom.2024.07.001.

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

T. Konstantin Rusch and Daniela Rus. Oscillatory state-space models. In *International Conference on Learning Representations*, 2025.

Louis Serrano, Thomas X. Wang, Etienne Le Naour, Jean-Noël Vittaut, and Patrick Gallinari. AROMA: Preserving spatial structure for latent PDE modeling with local neural fields. In *Advances in Neural Information Processing Systems*, 2024.

Ramansh Sharma and Varun Shankar. Ensemble and mixture-of-experts DeepONets for operator learning. *Transactions on Machine Learning Research*, 2025.

Hao Shen, Liting Yao, Xavier Legrand, and Peng Wang. Characterization of wrinkle morphologies by surface waviness evaluation method during deep forming of multilayer composite preforms. *Composite Structures*, 306:116586, 2023. doi: 10.1016/j.compstruct.2022.116586.

Vincent Sitzmann, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *Advances in Neural Information Processing Systems*, 2020.

T. Strouboulis, I. Babuška, and K. Copps. The design and analysis of the generalized finite element method. *Computer Methods in Applied Mechanics and Engineering*, 181(1–3):43–69, 2000. doi: 10.1016/S0045-7825(99)00072-9.

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

Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. In *Advances in Neural Information Processing Systems*, 2020.

Nathaniel Trask, Amelia Henriksen, Carianne Martinez, and Eric C. Cyr. Hierarchical partition of unity networks: Fast multilevel training. In *Proceedings of Mathematical and Scientific Machine Learning*, volume 190 of *Proceedings of Machine Learning Research*, pp. 271–286. PMLR, 2022.

Tian Wang and Chuang Wang. Latent neural operator for solving forward and inverse pde problems. In *Advances in Neural Information Processing Systems*, 2024.

Shizheng Wen, Arsh Kumbhat, Levi Lingsch, Sepehr Mousavi, Yizhou Zhao, Praveen Chandrashekar, and Siddhartha Mishra. Geometry aware operator transformer as an efficient and accurate neural surrogate for PDEs on arbitrary domains. In *Advances in Neural Information Processing Systems*, 2025.

Holger Wendland. Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. *Advances in Computational Mathematics*, 4:389–396, 1995. doi: 10.1007/BF02123482.

Holger Wendland. Fast evaluation of radial basis functions: Methods based on partition of unity. In Charles K. Chui, Larry L. Schumaker, and Joachim Stöckler (eds.), *Approximation Theory X: Wavelets, Splines, and Applications*, pp. 473–483. Vanderbilt University Press, Nashville, TN, 2002.

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

Tailin Wu, Takashi Maruyama, and Jure Leskovec. Learning to accelerate partial differential equations via latent global evolution. In *Advances in Neural Information Processing Systems*, 2022.

Tobias Würth, Niklas Freymuth, Gerhard Neumann, and Luise Kärger. Diffusion-based hierarchical graph neural networks for simulating nonlinear solid mechanics. In *Advances in Neural Information Processing Systems*, 2025.

Tobias Würth, Niklas Freymuth, Johannes Mitsch, Philipp Dahlinger, Tai Hoang, Gerhard Neumann, and Luise Kärger. Algebraic hierarchical graph neural networks for forming simulation of thermoplastic composite materials. *Key Engineering Materials*, 1050:31–42, 2026. doi: 10.4028/p-kah2tn.

Youn-Yeol Yu, Jeongwhan Choi, Woojin Cho, Kookjin Lee, Nayong Kim, Kiseok Chang, Changseung Woo, Ilho Kim, Seokwoo Lee, Joon Young Yang, Sooyoung Yoon, and Noseong Park. Learning flexible body collision dynamics with hierarchical contact mesh transformer. In *International Conference on Learning Representations*, 2024.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R. Salakhutdinov, and Alexander J. Smola. Deep sets. In *Advances in Neural Information Processing Systems*, 2017.

A Related Work on Partition-of-Unity Methods

Neural PDE simulation on general discretizations. Neural PDE simulators differ mainly in how they represent spatial interactions. FNO (Li et al., 2021) learns operators through spectral convolution, while CNO (Raonić et al., 2023) uses convolutional operators designed to remain consistent across discretizations. GINO (Li et al., 2023) combines graph operators on irregular geometries with a regular latent grid, whereas RIGNO (Mousavi et al., 2025) uses a multiscale regional graph. For time-dependent mesh simulation, MGN (Pfaff et al., 2021) evolves the state through local message passing, while IGNS (Hoang et al., 2026) improves long-range propagation using information-preserving dynamics. Transolver (Wu et al., 2024) and GAOT (Wen et al., 2025) instead use attention-based spatial representations for general geometries. Despite these differences, these methods generally evolve or repeatedly reconstruct a spatial field whose cost remains coupled to the physical resolution.

Latent dynamics for time-dependent PDEs. A separate line of work reduces rollout cost by evolving the dynamics in a compressed state. LE-PDE (Wu et al., 2022) learns a compact global latent representation, while UPT (Alkin et al., 2024) and LNO (Wang & Wang, 2024) map spatial observations to latent tokens before applying the learned operator. More recent approaches retain more spatial information, with AROMA (Serrano et al., 2024) combining spatially interpretable latent codes with local neural fields and CALM-PDE (Hagnberger et al., 2025) compressing the field onto learned latent query positions using local continuous convolutions. SPANS takes a different approach by organizing the latent state over fixed spatial patches, which can be defined independently of the input discretization. Our main distinction is an efficient latent representation that explicitly preserves localized fine-scale structure under strong spatial compression. To this end, SPANS combines spatially structured latent patch states, short- and long-range latent dynamics, and partition-of-unity blending over local neural fields. The classical development of partition-of-unity methods and related neural discretizations is discussed separately in Appendix A.

Partition-of-unity methods. The PUM constructs globally admissible approximation spaces by combining a partition of unity with independently chosen local approximation spaces (Melenk & Babuška, 1996; Babuška & Melenk, 1997). Its classical analysis establishes that global approximation quality can be controlled through local approximation errors, while allowing the local spaces to incorporate problem-specific structure such as boundary layers, singularities, or oscillatory solutions (Melenk & Babuška, 1997; Babuška et al., 2003). Closely related ideas appear in mesh-free hp-cloud methods (Duarte & Oden, 1996) and the generalized finite element method, which augments standard finite-element spaces with locally enriched functions through a partition of unity (Strouboulis et al., 2000). The same localization principle was subsequently developed for kernel approximations: RBF-PU methods combine local radial-basis expansions with partition-of-unity assembly to obtain scalable approximation on scattered points (Wendland, 2002), with later work establishing stable PDE discretizations and error estimates (Larsson et al., 2017; Mirzaei, 2021). Recent extensions treat time-dependent PDEs while retaining the localized spatial construction (Raeisi et al., 2024). Across these formulations, the spatial cover provides the global assembly mechanism while accuracy can be increased through richer local spaces rather than uniform refinement of the complete discretization, which is the classical principle underlying our patch-based formulation.

Neural partition-of-unity representations. Several recent approaches incorporate learned functions into related local-to-global constructions. POU-Nets (Lee et al., 2021) learn a mesh-free partition together with local polynomial approximation spaces, with subsequent probabilistic and hierarchical variants improving localization and multilevel approximation (Trask et al., 2022; Fan et al., 2023). Closely related FBPINNs (Moseley et al., 2023) place small neural networks on overlapping subdomains and assemble their predictions through smooth local windows; multilevel extensions introduce additional coarse decompositions to improve communication across many subdomains (Dolean et al., 2024). The N-adaptive Ritz method (Baek et al., 2024) instead uses neural networks to enrich partition-of-unity trial spaces within a variational PDE solver, while PoU-MoE DeepONets (Sharma & Shankar, 2025) localize neural-operator representations through smooth partition-of-unity weighting. These methods optimize a function or PDE solution representation, or learn an input–output operator. In contrast, SPANS brings the partition-of-unity principle into latent dynamics by associating a persistent latent state with each spatial patch, evolving these states

jointly over time, and using them to parameterize local neural representations that are smoothly combined when reconstructing the physical field.

B Proof of Theorem 1

We start recalling notation and assumptions. For a scalar field $v : \Omega \times [0, T] \to \mathbb{R}$, define its spatial $L^2(\Omega)$-norm at time t by

$$\|v(\cdot, t)\|_{L^2(\Omega)} := \left(\int_\Omega |v(\boldsymbol{x}, t)|^2 \, \mathrm{d}\boldsymbol{x} \right)^{1/2}. \tag{20}$$

The corresponding finite-horizon norm is:

$$\|v\|_{[0,T]} := \sup_{t \in [0,T]} \|v(\cdot, t)\|_{L^2(\Omega)}, \tag{21}$$

which measures the largest spatial $L^2(\Omega)$ magnitude of v over $[0, T]$.

For the spatial approximation, let $\{\Omega_p\}_{p=1}^P$ be an overlapping cover of Ω, with nonnegative weights satisfying $\mathrm{supp}(w_p) \subseteq \Omega_p$ and $\sum_{p=1}^P w_p(\boldsymbol{x}) = 1$. For each patch, define the local space:

$$V_p := \mathrm{span}\{b_{p,1}, \ldots, b_{p,J}\} \subset L^2(\Omega_p). \tag{22}$$

Then, we define the global PoU approximation space is:

$$V_{\mathrm{PU}} := \left\{ \sum_{p=1}^P w_p v_p : v_p \in V_p \right\} = \mathrm{span}\left\{ w_p b_{p,j},\ p = 1, \ldots, P, j = 1, \ldots, J \right\}. \tag{23}$$

Then, let $\boldsymbol{Z}^\star = [\boldsymbol{z}_1^\star, \ldots, \boldsymbol{z}_P^\star]^\top$ and $\boldsymbol{Z} = [\boldsymbol{z}_1, \ldots, \boldsymbol{z}_P]^\top$ in $\mathbb{R}^{P \times J}$ denote the reference and learned structured latent states, respectively. Similar to above, we define their vectorized forms as in Section 3:

$$\boldsymbol{z}^\star(t) := \mathrm{vec}\big(\boldsymbol{Z}^\star(t)\big), \qquad \boldsymbol{z}(t) := \mathrm{vec}\big(\boldsymbol{Z}(t)\big) \in \mathbb{R}^{PJ}. \tag{24}$$

Thus, $\|\boldsymbol{z}^\star - \boldsymbol{z}\|_2^2 = \sum_{p=1}^P \big\|\boldsymbol{z}_p^\star - \boldsymbol{z}_p\big\|_2^2$. Consistently with Eq. (12), we also define $\boldsymbol{b}_p(\boldsymbol{x}) = [b_{p,1}(\boldsymbol{x}), \ldots, b_{p,J}(\boldsymbol{x})]^\top$, such that the reference field and its decoded output are given by

$$y^\star(\boldsymbol{x}, t) := \sum_{p=1}^P w_p(\boldsymbol{x}) \boldsymbol{b}_p^\top(\boldsymbol{x}) \boldsymbol{z}_p^\star(t), \tag{25}$$

$$\widetilde{u}^\star(\boldsymbol{x}, t) := h_{\theta_{\mathrm{rec}}}\big(\boldsymbol{x}, y^\star(\boldsymbol{x}, t)\big). \tag{26}$$

The learned quantities y and $\widetilde{u}$ are defined analogously from $\boldsymbol{z}$.

Assumption 1 (Galerkin stability)**.** *The Galerkin approximation of the underlying PDE with solution u is stable: there exists a scalar $B_T \geq 0$, such that the projected solution, $y^\star \in C([0, T]; V_{\mathrm{PU}})$ satisfies*

$$\|u - y^\star\|_{[0,T]} \leq B_T \inf_{y \in C([0,T]; V_{\mathrm{PU}})} \|u - y\|_{[0,T]}. \tag{27}$$

Assumption 1 requires the Galerkin approximation of the underlying PDE to be stable, in the sense that its error remains within a factor B_T of the best approximation available in V_{PU}. This condition holds for broad classes of well-posed PDEs under standard Galerkin conditions, as established in classical Galerkin and PUM theory (Melenk & Babuška, 1996; Babuška & Melenk, 1997). The stability estimate is determined by the underlying PDE formulation, while the PUM construction assembles the local approximation spaces into the global field. Consequently, the same analysis extends to further PDE classes whenever their corresponding Galerkin stability estimate is available. The subscript T makes explicit that the stability factor may depend on the finite time horizon.

Assumption 2 (Latent-dynamics stability)**.** *The reference and learned projected trajectories satisfy*

$$\dot{\boldsymbol{z}}^\star(t) = f\big(\boldsymbol{z}^\star(t)\big), \qquad \dot{\boldsymbol{z}}(t) = f_{\theta_{\mathrm{dyn}}}\big(\boldsymbol{z}(t)\big). \tag{28}$$

There exists a compact set $\mathcal{K} \subset \mathbb{R}^{PJ}$ such that $\boldsymbol{z}^\star(t), \boldsymbol{z}(t) \in \mathcal{K}$ for all $t \in [0, T]$. Moreover, $f_{\theta_{\mathrm{dyn}}}$ is L_{dyn}-Lipschitz on $\mathcal{K}$, meaning that

$$\big\|f_{\theta_{\mathrm{dyn}}}(\boldsymbol{\zeta}_1) - f_{\theta_{\mathrm{dyn}}}(\boldsymbol{\zeta}_2)\big\|_2 \leq L_{\mathrm{dyn}} \|\boldsymbol{\zeta}_1 - \boldsymbol{\zeta}_2\|_2, \quad \forall \boldsymbol{\zeta}_1, \boldsymbol{\zeta}_2 \in \mathcal{K}. \tag{29}$$

Finally, we define the encoding and dynamics approximation errors by

$$\varepsilon_{\mathrm{enc}} := \|\boldsymbol{z}^\star(0) - \boldsymbol{z}(0)\|_2 , \tag{30}$$

$$\varepsilon_{\mathrm{dyn}} := \sup_{t\in[0,T]} \left\| f\big(\boldsymbol{z}^\star(t)\big) - f_{\theta_{\mathrm{dyn}}}\big(\boldsymbol{z}^\star(t)\big) \right\|_2 . \tag{31}$$

Assumption 3 (Decoder smoothness & expressivity)**.** *$h_{\theta_{\mathrm{rec}}}$ is uniformly L_{rec}-Lipschitz in its second argument:*

$$|h_{\theta_{\mathrm{rec}}}(\boldsymbol{x}, y_a) - h_{\theta_{\mathrm{rec}}}(\boldsymbol{x}, y_b)| \le L_{\mathrm{rec}} |y_a - y_b| , \quad \forall y_a, y_b \in V_{\mathrm{PU}}. \tag{32}$$

For every $\varepsilon_{\mathrm{dec}} > 0$, there exists a decoder such that its reconstruction error along the reference field satisfies:

$$\|y^\star - \widetilde{u}^\star\|_{[0,T]} \le \varepsilon_{\mathrm{dec}}. \tag{33}$$

We need two lemmas before proving our main result.

Lemma 1 (Galerkin–PUM approximation)**.** *Let u be the solution to the PDE. The global approximation error can be bound in terms of the local approximations $\{q_p \in C([0,T]; V_p)\}_{p=1}^P$:*

$$\|u - y^\star\|^2_{[0,T]} \le B_T^2 \varepsilon^2_{\mathrm{glo}}, \tag{34}$$

where

$$\varepsilon^2_{\mathrm{glo}} := \sum_{p=1}^{P} \sup_{t\in[0,T]} \|u(\cdot,t) - q_p(\cdot,t)\|^2_{L^2(\Omega_p)} . \tag{35}$$

Proof. From Assumption 1, we just have to show the following:

$$\inf_{y'\in C([0,T];V_{\mathrm{PU}})} \|u - y'\|^2_{[0,T]} \le \varepsilon^2_{\mathrm{glo}}. \tag{36}$$

For any $y \in C([0,T]; V_{\mathrm{PU}})$, we have:

$$\inf_{y'\in C([0,T];V_{\mathrm{PU}})} \|u - y'\|^2_{[0,T]} \le \|u - y\|^2_{[0,T]} . \tag{37}$$

Let us define the PoU combination of the local approximants q_p as:

$$y(\boldsymbol{x}, t) := \sum_{p=1}^{P} w_p(\boldsymbol{x}) q_p(\boldsymbol{x}, t) \in V_{\mathrm{PU}}. \tag{38}$$

Since the weights form a convex combination, Jensen's inequality gives

$$\begin{aligned} |u(\boldsymbol{x},t) - y(\boldsymbol{x},t)|^2 &= \left| u(\boldsymbol{x},t) - \sum_{p=1}^{P} w_p(\boldsymbol{x}) q_p(\boldsymbol{x},t) \right|^2 \\ &= \left| \sum_{p=1}^{P} w_p(\boldsymbol{x}) \big(u(\boldsymbol{x},t) - q_p(\boldsymbol{x},t)\big) \right|^2 \\ &\le \sum_{p=1}^{P} w_p(\boldsymbol{x}) \, |u(\boldsymbol{x},t) - q_p(\boldsymbol{x},t)|^2 . \end{aligned} \tag{39}$$

Expanding norm definitions Eq. (21) and Eq. (20), and using inequality (39), where the supremum applies point-wise on $[0, T]$, we have:

$$\begin{aligned}
\|u - y\|^2_{[0,T]} &= \sup_{t\in[0,T]} \|u(\cdot,t) - y(\cdot,t)\|^2_{L^2(\Omega)} \\
&= \sup_{t\in[0,T]} \int_\Omega |u(\boldsymbol{x},t) - y(\boldsymbol{x},t)|^2 \,\mathrm{d}\boldsymbol{x} \\
&\leq \sup_{t\in[0,T]} \int_\Omega \sum_{p=1}^{P} w_p(\boldsymbol{x}) \,|u(\boldsymbol{x},t) - q_p(\boldsymbol{x},t)|^2 \,\mathrm{d}\boldsymbol{x} \\
&\leq \sum_{p=1}^{P} \sup_{t\in[0,T]} \int_{\Omega_p} w_p(\boldsymbol{x}) \,|u(\boldsymbol{x},t) - q_p(\boldsymbol{x},t)|^2 \,\mathrm{d}\boldsymbol{x} \\
&\leq \sum_{p=1}^{P} \sup_{t\in[0,T]} \int_{\Omega_p} |u(\boldsymbol{x},t) - q_p(\boldsymbol{x},t)|^2 \,\mathrm{d}\boldsymbol{x}, \qquad (40)
\end{aligned}$$

where the final inequality uses $0 \leq w_p \leq 1$. Combining the assumption (27), with steps (37) and (40), and with definitions (35) and (20) concludes the proof:

$$\begin{aligned}
\|u - y^\star\|^2_{[0,T]} &\leq B_T^2 \inf_{y'\in C([0,T];V_{\mathrm{PU}})} \|u - y'\|^2_{[0,T]} \\
&\leq B_T^2 \,\|u - y\|^2_{[0,T]} \\
&\leq B_T^2 \sum_{p=1}^{P} \sup_{t\in[0,T]} \int_{\Omega_p} |u(\boldsymbol{x},t) - q_p(\boldsymbol{x},t)|^2 \,\mathrm{d}\boldsymbol{x} \\
&= B_T^2 \sum_{p=1}^{P} \sup_{t\in[0,T]} \|u(\cdot,t) - q_p(\cdot,t)\|^2_{L^2(\Omega_p)} \\
&= B_T^2 \varepsilon^2_{\mathrm{glo}}. \qquad (41)
\end{aligned}$$

□

This lemma shows that, provided the Galerkin approximation of the underlying PDE is stable in the sense of Assumption 1, the global reconstruction error is controlled by the local approximation errors. Consequently, improving the local approximation spaces also improves the global field approximation. We next quantify how the encoding and dynamics approximation errors propagate along the latent trajectory.

Lemma 2 (Stability of learned coefficient dynamics)**.** *Let $\boldsymbol{z}^\star$ and $\boldsymbol{z}$ denote the reference and learned projected solutions, respectively. Suppose that Assumption 2 holds and that the errors $\varepsilon_{\mathrm{enc}}$ (Eq.* (30)*) and $\varepsilon_{\mathrm{dyn}}$ (Eq.* (31)*) are finite, then:*

$$\sup_{t\in[0,T]} \|\boldsymbol{z}^\star(t) - \boldsymbol{z}(t)\|_2 \leq e^{L_{\mathrm{dyn}}T} \left(\varepsilon_{\mathrm{enc}} + T\varepsilon_{\mathrm{dyn}}\right). \qquad (42)$$

Proof. Define $e(t) := \|\boldsymbol{z}^\star(t) - \boldsymbol{z}(t)\|_2$ and integrate both sides of Eq. (28) yields:

$$
\begin{aligned}
e(t) &= \left\| \boldsymbol{z}^\star(0) + \int_0^t f(\boldsymbol{z}^\star(s))\mathrm{d}s - \left(\boldsymbol{z}(0) + \int_0^t f_{\theta_{\mathrm{dyn}}}(\boldsymbol{z}(s))\mathrm{d}s \right) \right\|_2 \\
&= \left\| \boldsymbol{z}^\star(0) - \boldsymbol{z}(0) + \int_0^t \left(f(\boldsymbol{z}^\star(s)) - f_{\theta_{\mathrm{dyn}}}(\boldsymbol{z}(s)) \right) \mathrm{d}s \right\|_2 \qquad (43) \\
&\leq \|\boldsymbol{z}^\star(0) - \boldsymbol{z}(0)\| + \left\| \int_0^t \left(f(\boldsymbol{z}^\star(s)) - f_{\theta_{\mathrm{dyn}}}(\boldsymbol{z}(s)) \right) \mathrm{d}s \right\|_2 \\
&= \varepsilon_{\mathrm{enc}} + \left\| \int_0^t f(\boldsymbol{z}^\star(s)) - f_{\theta_{\mathrm{dyn}}}(\boldsymbol{z}(s))\mathrm{d}s \right\|_2 \\
&\leq \varepsilon_{\mathrm{enc}} + \int_0^t \left\| f(\boldsymbol{z}^\star(s)) - f_{\theta_{\mathrm{dyn}}}(\boldsymbol{z}(s)) \right\|_2 \mathrm{d}s \\
&= \varepsilon_{\mathrm{enc}} + \int_0^t \left(\left\| f(\boldsymbol{z}^\star(s)) - f_{\theta_{\mathrm{dyn}}}(\boldsymbol{z}^\star(s)) + f_{\theta_{\mathrm{dyn}}}(\boldsymbol{z}^\star(s)) - f_{\theta_{\mathrm{dyn}}}(\boldsymbol{z}(s)) \right\|_2 \right) \mathrm{d}s \\
&\leq \varepsilon_{\mathrm{enc}} + \int_0^t \left(\left\| f(\boldsymbol{z}^\star(s)) - f_{\theta_{\mathrm{dyn}}}(\boldsymbol{z}^\star(s)) \right\|_2 + \left\| f_{\theta_{\mathrm{dyn}}}(\boldsymbol{z}^\star(s)) - f_{\theta_{\mathrm{dyn}}}(\boldsymbol{z}(s)) \right\|_2 \right) \mathrm{d}s \\
&= \varepsilon_{\mathrm{enc}} + \int_0^t \left\| f(\boldsymbol{z}^\star(s)) - f_{\theta_{\mathrm{dyn}}}(\boldsymbol{z}^\star(s)) \right\|_2 \mathrm{d}s + \int_0^t \left\| f_{\theta_{\mathrm{dyn}}}(\boldsymbol{z}^\star(s)) - f_{\theta_{\mathrm{dyn}}}(\boldsymbol{z}(s)) \right\|_2 \mathrm{d}s \\
&\leq \varepsilon_{\mathrm{enc}} + t\varepsilon_{\mathrm{dyn}} + L_{\mathrm{dyn}} \int_0^t e(s)\mathrm{d}s, \quad \forall t \in [0, T], \qquad (44)
\end{aligned}
$$

where the first three inequalities follow from the triangle inequality. The fourth inequality comes from two facts: bounding the definition Eq. (31), and Assumption 2:

$$
\begin{aligned}
\int_0^t \left\| f(\boldsymbol{z}^\star(s)) - f_{\theta_{\mathrm{dyn}}}(\boldsymbol{z}^\star(s)) \right\|_2 \mathrm{d}s &\leq \int_0^t \sup_{t \in [0,T]} \left\| f\big(\boldsymbol{z}^\star(t)\big) - f_{\theta_{\mathrm{dyn}}}\big(\boldsymbol{z}^\star(t)\big) \right\|_2 \mathrm{d}s \\
&\leq t \sup_{t \in [0,T]} \left\| f\big(\boldsymbol{z}^\star(t)\big) - f_{\theta_{\mathrm{dyn}}}\big(\boldsymbol{z}^\star(t)\big) \right\|_2 \\
&= t\varepsilon_{\mathrm{dyn}}, \qquad (45) \\
\int_0^t \left\| f_{\theta_{\mathrm{dyn}}}(\boldsymbol{z}^\star(s)) - f_{\theta_{\mathrm{dyn}}}(\boldsymbol{z}(s)) \right\|_2 \mathrm{d}s &\leq \int_0^t L_{\mathrm{dyn}} \left\| \boldsymbol{z}^\star(s) - \boldsymbol{z}(s) \right\|_2 \mathrm{d}s \\
&= L_{\mathrm{dyn}} \int_0^t e(s)\mathrm{d}s. \qquad (46)
\end{aligned}
$$

We want to bound (44) further. Since e is nonnegative, $\varepsilon_{\mathrm{enc}} + t\varepsilon_{\mathrm{dyn}}$ is nondecreasing, and the integral form of Grönwall's inequality yields

$$
\begin{aligned}
e(t) &\leq \varepsilon_{\mathrm{enc}} + t\varepsilon_{\mathrm{dyn}} + L_{\mathrm{dyn}} \int_0^t e(s)\mathrm{d}s \\
&\leq \left(\varepsilon_{\mathrm{enc}} + t\varepsilon_{\mathrm{dyn}} \right) e^{L_{\mathrm{dyn}} t}, \quad \forall t \in [0, T]. \qquad (47)
\end{aligned}
$$

Since from Eq. (31) we know that $\varepsilon_{\mathrm{dyn}} \geq 0$, taking the supremum over $t \in [0, T]$ concludes the proof:

$$
\begin{aligned}
\sup_{t \in [0,T]} e(t) &\leq \sup_{t \in [0,T]} \left(\varepsilon_{\mathrm{enc}} + t\varepsilon_{\mathrm{dyn}} \right) e^{L_{\mathrm{dyn}} t} \\
&\leq \left(\varepsilon_{\mathrm{enc}} + T\varepsilon_{\mathrm{dyn}} \right) e^{L_{\mathrm{dyn}} T}. \qquad (48)
\end{aligned}
$$

□

Together, the two lemmas and the decoder assumption yield the following approximation error for the whole framework.

Let's recall some definitions. Let $\widetilde{u}$ be the reconstructed field as defined in Eq. (12). Let $\varepsilon_{\mathrm{dec}}$, $\varepsilon_{\mathrm{dyn}}$, $\varepsilon_{\mathrm{enc}}$, $\varepsilon_{\mathrm{glo}}$, B_T, and L_{rec} be the constants defined earlier in this section. And define the coefficient-to-field constant for the fixed cover and local spaces:

$$C_b^2 := \max_{1\le p\le P} \int_\Omega w_p(\boldsymbol{x}) \left\|\boldsymbol{b}_p(\boldsymbol{x})\right\|_2^2 \mathrm{d}\boldsymbol{x}. \tag{49}$$

Theorem 1 (Approximation Error)**.** *Let $u \in C([0,T]; L^2(\Omega))$ be a solution to Eq.* (1) *on a bounded domain $\Omega \subset \mathbb{R}^d$. Suppose that Assumptions 1 to 3 hold and that $\varepsilon_{\mathrm{enc}}$ and $\varepsilon_{\mathrm{dyn}}$ are finite. Then:*

$$\left\|u - \widetilde{u}\right\|_{[0,T]} \le B_T\varepsilon_{\mathrm{glo}} + \varepsilon_{\mathrm{dec}} + L_{\mathrm{rec}} C_b e^{L_{\mathrm{dyn}}T} \left(\varepsilon_{\mathrm{enc}} + T\varepsilon_{\mathrm{dyn}}\right) \tag{50}$$

Proof. By adding and subtracting $y^\star$ and $\widetilde{u}^\star$, and applying the triangle inequality, we have:

$$\begin{aligned}\left\|u - \widetilde{u}\right\|_{[0,T]} &\le \left\|u - y^\star\right\|_{[0,T]} + \left\|y^\star - \widetilde{u}^\star\right\|_{[0,T]} + \left\|\widetilde{u}^\star - \widetilde{u}\right\|_{[0,T]} \\ &\le B_T\varepsilon_{\mathrm{glo}} + \varepsilon_{\mathrm{dec}} + \left\|\widetilde{u}^\star - \widetilde{u}\right\|_{[0,T]}, \end{aligned} \tag{51}$$

where the first two terms are given by Lemma 1 and the decoder expressivity from Assumption 3, respectively. To bound the final term, we use the decoder smoothness from Assumption 3:

$$\left\|\widetilde{u}^\star - \widetilde{u}\right\|_{[0,T]} \le L_{\mathrm{rec}} \left\|y^\star - y\right\|_{[0,T]}. \tag{52}$$

Recall from (20)–(21):

$$\begin{aligned}\left\|y^\star - y\right\|_{[0,T]} &= \sup_{t\in[0,T]} \left\|y^\star(\cdot,t) - y(\cdot,t)\right\|_{L^2(\Omega)} \\ &= \sup_{t\in[0,T]} \left(\int_\Omega \left|y^\star(\boldsymbol{x},t) - y(\boldsymbol{x},t)\right|^2 \mathrm{d}\boldsymbol{x}\right)^{1/2}. \end{aligned} \tag{53}$$

For fixed $(\boldsymbol{x}, t)$, Jensen's inequality for the partition-of-unity weights from Eq. (25), followed by the Cauchy–Schwarz inequality, gives

$$\begin{aligned}\left|y^\star(\boldsymbol{x},t) - y(\boldsymbol{x},t)\right|^2 &= \left|\sum_{p=1}^P w_p(\boldsymbol{x}) \boldsymbol{b}_p^\top(\boldsymbol{x}) \big(\boldsymbol{z}_p^\star(t) - \boldsymbol{z}_p(t)\big)\right|^2 \\ &\le \sum_{p=1}^P w_p(\boldsymbol{x}) \left|\boldsymbol{b}_p^\top(\boldsymbol{x}) \big(\boldsymbol{z}_p^\star(t) - \boldsymbol{z}_p(t)\big)\right|^2 \\ &\le \sum_{p=1}^P w_p(\boldsymbol{x}) \left\|\boldsymbol{b}_p(\boldsymbol{x})\right\|_2^2 \left\|\boldsymbol{z}_p^\star(t) - \boldsymbol{z}_p(t)\right\|_2^2. \end{aligned} \tag{54}$$

Integrating both sides of the inequality (54) over Ω gives:

$$\begin{aligned}\int_\Omega \left|y^\star(\boldsymbol{x},t) - y(\boldsymbol{x},t)\right|^2 \mathrm{d}\boldsymbol{x} &= \left\|y^\star(\cdot,t) - y(\cdot,t)\right\|_{L^2(\Omega)}^2 \\ &\le \int_\Omega \sum_{p=1}^P w_p(\boldsymbol{x}) \left\|\boldsymbol{b}_p(\boldsymbol{x})\right\|_2^2 \left\|\boldsymbol{z}_p^\star(t) - \boldsymbol{z}_p(t)\right\|_2^2 \mathrm{d}\boldsymbol{x} \\ &= \sum_{p=1}^P \left(\left(\int_\Omega w_p(\boldsymbol{x}) \left\|\boldsymbol{b}_p(\boldsymbol{x})\right\|_2^2 \mathrm{d}\boldsymbol{x}\right) \left\|\boldsymbol{z}_p^\star(t) - \boldsymbol{z}_p(t)\right\|_2^2\right) \\ &\le \sum_{p=1}^P \left(\max_{1\le p\le P} \int_\Omega w_p(\boldsymbol{x}) \left\|\boldsymbol{b}_p(\boldsymbol{x})\right\|_2^2 \mathrm{d}\boldsymbol{x} \left\|\boldsymbol{z}_p^\star(t) - \boldsymbol{z}_p(t)\right\|_2^2\right) \\ &= \max_{1\le p\le P} \int_\Omega w_p(\boldsymbol{x}) \left\|\boldsymbol{b}_p(\boldsymbol{x})\right\|_2^2 \mathrm{d}\boldsymbol{x} \sum_{p=1}^P \left\|\boldsymbol{z}_p^\star(t) - \boldsymbol{z}_p(t)\right\|_2^2 \\ &= C_b^2 \left\|\boldsymbol{z}^\star(t) - \boldsymbol{z}(t)\right\|_2^2. \end{aligned} \tag{55}$$

Taking the square root and then the supremum on both sides of Eq. (55), and using Lemma 2, we have:

$$\begin{aligned}\sup_{t\in[0,T]} \|y^\star(\cdot,t) - y(\cdot,t)\|_{L^2(\Omega)} &= \|y^\star - y\|_{[0,T]} \\ &\le \sup_{t\in[0,T]} C_b \left\|\boldsymbol{z}^\star(t) - \boldsymbol{z}(t)\right\|_2 \\ &= C_b \sup_{t\in[0,T]} \left\|\boldsymbol{z}^\star(t) - \boldsymbol{z}(t)\right\|_2 \\ &\le C_b e^{L_{\mathrm{dyn}}T} \left(\varepsilon_{\mathrm{enc}} + T\varepsilon_{\mathrm{dyn}}\right). \end{aligned} \tag{56}$$

Using the bound (56) in (52) yields

$$\|\widetilde{u}^\star - \widetilde{u}\|_{[0,T]} \le L_{\mathrm{rec}} C_b e^{L_{\mathrm{dyn}}T} \left(\varepsilon_{\mathrm{enc}} + T\varepsilon_{\mathrm{dyn}}\right). \tag{57}$$

Using (57) in (51) concludes the proof. □

C SPANS IMPLEMENTATION DETAILS

This section gives the implementation details that the main text defers: the processor (Appendix C.1), the decoder (Appendix C.2), and sampled-query training (Appendix C.3).

C.1 PROCESSOR

Processor input. At every step t, the processor reads one token per patch,

$$\left[\boldsymbol{z}_p(t), \mathrm{RFF}(\boldsymbol{c}_p)\right], \qquad \mathrm{RFF}(\boldsymbol{c}_p) = \left[\sin(2\pi \boldsymbol{F}^\top \boldsymbol{c}_p), \cos(2\pi \boldsymbol{F}^\top \boldsymbol{c}_p)\right], \tag{58}$$

which a linear layer maps to the hidden width before the attention blocks. $\mathrm{RFF}(\cdot)$ are random Fourier features of the patch center (Tancik et al., 2020), whose frequency matrix $\boldsymbol{F} \in \mathbb{R}^{d\times 32}$ is drawn from $\mathcal{N}(0, 6^2)$ once at initialization and kept fixed. The centers do not change during rollout, hence $\mathrm{RFF}(\boldsymbol{c}_p)$ is computed once and reused at every step; only $\boldsymbol{z}_p(t)$ changes. For second-order systems, $\boldsymbol{z}_p(t)$ is replaced by the pair $[\boldsymbol{z}_p(t), \boldsymbol{v}_p(t)]$, where $\boldsymbol{v}_p(t)$ is the p-th row of the latent velocity $\boldsymbol{V}_t$ in Eq. (60).

First-order and second-order updates. For first-order systems, the processor predicts the latent displacement $\Delta \boldsymbol{Z}_t = \mathrm{Attn}_{\theta_{\mathrm{short}}}(\boldsymbol{Z}_t) + \mathrm{Attn}_{\theta_{\mathrm{long}}}(\boldsymbol{Z}_t)$ and applies the residual update in Eq. (16). The deformable tasks (FormNapf, FormDome, and Teddy) instead evolve the pair $(\boldsymbol{Z}_t, \boldsymbol{V}_t)$, where $\boldsymbol{V}_t \in \mathbb{R}^{P\times J}$ is the displacement in latent coefficients from the previous time step $\boldsymbol{Z}_t - \boldsymbol{Z}_{t-1}$. The supplied initial position and velocity determine the preceding and current position fields, which are encoded separately as $\boldsymbol{Z}_{-1}$ and $\boldsymbol{Z}_0$. We initialize $\boldsymbol{V}_0 = \boldsymbol{Z}_0 - \boldsymbol{Z}_{-1}$. From the feature-wise concatenation $[\boldsymbol{Z}_t, \boldsymbol{V}_t]$, the processor predicts the latent acceleration

$$\Delta \boldsymbol{V}_t = \mathrm{Attn}_{\theta_{\mathrm{short}}}\big([\boldsymbol{Z}_t, \boldsymbol{V}_t]\big) + \mathrm{Attn}_{\theta_{\mathrm{long}}}\big([\boldsymbol{Z}_t, \boldsymbol{V}_t]\big) + \mathbf{1}_P \boldsymbol{\beta}_{\mathrm{acc}}{}^\top, \tag{59}$$

where $\boldsymbol{\beta}_{\mathrm{acc}} \in \mathbb{R}^J$ is a learned, trajectory-independent acceleration. We then apply the semi-implicit Euler update (Hairer et al., 2006; Rusch & Rus, 2025) with the learned acceleration $\Delta \boldsymbol{V}_t$ as follows:

$$\boldsymbol{V}_{t+1} = \boldsymbol{V}_t + \Delta_t\, \Delta \boldsymbol{V}_t, \qquad \boldsymbol{Z}_{t+1} = \boldsymbol{Z}_t + \Delta_t\, \boldsymbol{V}_{t+1}. \tag{60}$$

We set $\Delta_t = 1$ because the data use a fixed sampling interval and the learned acceleration absorbs this scale.

Action conditioning. The forming tasks provide only an initial action $\boldsymbol{a}_0 \in \mathbb{R}^{d_a}$. Its entries specify the gripper positions and pull magnitudes for the trajectory. We standardize each channel of $\boldsymbol{a}_0$ using the training split and reorder the entries into the canonical form [positions, pulls]. The map $g_{\mathrm{act}} : \mathbb{R}^{d_a} \to \mathbb{R}^J$ is an MLP. It produces one vector with the same width J as a patch state. We add this vector to every initial patch state,

$$\boldsymbol{Z}_0 \leftarrow \boldsymbol{Z}_0 + \mathbf{1}_P g_{\mathrm{act}}(\boldsymbol{a}_0)^\top. \tag{61}$$

Here $\boldsymbol{Z}_0 \in \mathbb{R}^{P\times J}$ contains the initial patch states and $\mathbf{1}_P \in \mathbb{R}^P$ is the all-ones vector. The outer product therefore broadcasts the same action embedding to all P patches. It then conditions the full rollout through the latent state without increasing its dimension.

C.2 DECODER

Wendland weights. Let $\chi(r) = (1-r)_+^4(4r+1)$, where $(1-r)_+ = \max(1-r, 0)$. The unnormalized and normalized weights are defined as

$$\widetilde{w}_p(\boldsymbol{x}) = \chi\left(\|\boldsymbol{x} - \boldsymbol{c}_p\|_2 / \rho\right), \qquad w_p(\boldsymbol{x}) = \widetilde{w}_p(\boldsymbol{x}) \Big/ \sum_{p'=1}^{P} \widetilde{w}_{p'}(\boldsymbol{x}), \tag{62}$$

where ρ denotes the support radius and $\boldsymbol{c}_p$ is the center of patch p associated with the query point $\boldsymbol{x}$ (see, e.g., (Wendland, 1995)).

Coordinate features. The decoder and its hypernetwork receive coordinates through one fixed Fourier feature map (Tancik et al., 2020). For a coordinate $\boldsymbol{x} \in \mathbb{R}^d$, the map concatenates $\boldsymbol{x}$ with sinusoids of a normalized coordinate $\bar{\boldsymbol{x}}$,

$$\gamma(\boldsymbol{x}) = \big[\boldsymbol{x}, \sin(\pi\bar{\boldsymbol{x}}), \cos(\pi\bar{\boldsymbol{x}}), \ldots, \sin(k\pi\bar{\boldsymbol{x}}), \cos(k\pi\bar{\boldsymbol{x}})\big], \tag{63}$$

where we set $k = 4$, so that $\gamma(\boldsymbol{x}) \in \mathbb{R}^{d_\gamma}$ with $d_\gamma = (2k+1)d$. For a query point, $\bar{\boldsymbol{x}}$ is its coordinate in the frame of its nearest patch, which is obtained by subtracting the patch center and dividing each coordinate by the corresponding bounding-box half-width. This ensures that $\bar{\boldsymbol{x}}$ is approximately in $[-1, 1]^d$ within the patch. For a patch center $\boldsymbol{c}_p$, which enters the hypernetwork in Eq. (64), no such normalization is applied, i.e., $\bar{\boldsymbol{x}} = \boldsymbol{x} = \boldsymbol{c}_p$.

Generated layer matrices. The decoder is a single MLP with $L = 4$ layers: three hidden layers of width $D = 128$ with SiLU activations, and a linear output layer. Layer ℓ applies a matrix $\boldsymbol{A}^{(\ell)} \in \mathbb{R}^{D_\ell \times D_{\ell-1}}$ generated by the hypernetwork $\mathcal{H}_\phi$, where $D_0 = J + d_\gamma$, $D_1 = D_2 = D_3 = D$, and D_L is the number of predicted channels. We adapt the low-rank parameterization introduced in IGNS for time-varying Hamiltonian matrices (Hoang et al., 2026), which combines fixed shared bases with a small set of generated coefficients. The essential difference is what these coefficients depend on. Unlike IGNS, where the weight matrices are functions of a time embedding alone and can therefore be precomputed for a rollout, here they depend on the evolving patch states and must be regenerated at every step. For each patch p, the hypernetwork generates the coefficient vectors of all layers,

$$\big\{\boldsymbol{d}_p^{(\ell)}(t),\, \widetilde{\boldsymbol{d}}_p^{(\ell)}(t)\big\}_{\ell=1}^{L} = \mathcal{H}_\phi\big(\boldsymbol{z}_p(t), \gamma(\boldsymbol{c}_p)\big), \qquad \boldsymbol{d}_p^{(\ell)}(t),\, \widetilde{\boldsymbol{d}}_p^{(\ell)}(t) \in \mathbb{R}^r, \tag{64}$$

and the matrix of layer ℓ is

$$\begin{aligned} \boldsymbol{A}_p^{(\ell)}(t) &= \boldsymbol{\Phi}^{(\ell)} \operatorname{diag}\big(\boldsymbol{d}_p^{(\ell)}(t)\big) \boldsymbol{\Psi}^{(\ell)\top} + \boldsymbol{R}_p^{(\ell)}(t) - \boldsymbol{R}_p^{(\ell)}(t)^\top, \\ \boldsymbol{R}_p^{(\ell)}(t) &= \widetilde{\boldsymbol{\Phi}}^{(\ell)} \operatorname{diag}\big(\widetilde{\boldsymbol{d}}_p^{(\ell)}(t)\big) \widetilde{\boldsymbol{\Psi}}^{(\ell)\top}, \end{aligned} \tag{65}$$

with bases $\boldsymbol{\Phi}^{(\ell)} \in \mathbb{R}^{D_\ell \times r}$ and $\boldsymbol{\Psi}^{(\ell)} \in \mathbb{R}^{D_{\ell-1} \times r}$, and likewise for the tilded bases. The bases are shared across patches and constant in time; only the coefficients depend on the evolving patch state. The antisymmetric term $\boldsymbol{R}_p^{(\ell)} - \boldsymbol{R}_p^{(\ell)\top}$ is added only to the square hidden layers $\ell = 2, 3$. At a query $\boldsymbol{x}$, the matrices are blended with the partition-of-unity weights,

$$\boldsymbol{A}^{(\ell)}(\boldsymbol{x}, t) := \sum_{p=1}^{P} w_p(\boldsymbol{x})\, \boldsymbol{A}_p^{(\ell)}(t). \tag{66}$$

Since Eq. (65) is linear in the coefficients and its bases do not depend on p, this equals Eq. (65) evaluated once at the blended coefficients $\sum_p w_p(\boldsymbol{x})\, \boldsymbol{d}_p^{(\ell)}(t)$ and $\sum_p w_p(\boldsymbol{x})\, \widetilde{\boldsymbol{d}}_p^{(\ell)}(t)$, so the matrices $\boldsymbol{A}_p^{(\ell)}(t)$ are never formed, i.e.,

$$\boldsymbol{A}^{(\ell)}(\boldsymbol{x}, t) = \boldsymbol{\Phi}^{(\ell)} \operatorname{diag}\left(\sum_{p=1}^{P} w_p(\boldsymbol{x})\, \boldsymbol{d}_p^{(\ell)}(t)\right) \boldsymbol{\Psi}^{(\ell)\top} + \boldsymbol{R}^{(\ell)}(\boldsymbol{x}, t) - \boldsymbol{R}^{(\ell)}(\boldsymbol{x}, t)^\top. \tag{67}$$

We use rank $r = 128$ and a two-hidden-layer hypernetwork of width 256 with SiLU activations.

Layerwise decoder. We finally describe the overall decoder architecture. Note that in Section 5 for the sake of simplicity, we suppress the coordinate features and use $\boldsymbol{A}^{(1)}$ to denote the first-layer block acting on the blended latent state. In the complete implementation, the first-layer input is $[\tilde{\boldsymbol{z}}(\boldsymbol{x},t), \gamma(\boldsymbol{x})]$, and the weight matrix of the first layer $\boldsymbol{A}^{(1)}(\boldsymbol{x},t)$ splits into a latent block $\boldsymbol{B}(\boldsymbol{x},t)$ and a coordinate block $\boldsymbol{C}(\boldsymbol{x},t)$ as follows:

$$\boldsymbol{A}^{(1)}(\boldsymbol{x},t) = \big[\, \boldsymbol{B}(\boldsymbol{x},t) \;\; \boldsymbol{C}(\boldsymbol{x},t) \,\big], \qquad \boldsymbol{B}(\boldsymbol{x},t) \in \mathbb{R}^{D\times J}, \quad \boldsymbol{C}(\boldsymbol{x},t) \in \mathbb{R}^{D\times d_\gamma}. \tag{68}$$

The block $\boldsymbol{B}$ is the reconstruction weight, and the remaining computation is the composition of MLPs formed by $h_{\theta_{\mathrm{rec}}}(\boldsymbol{x},\boldsymbol{y})$:

$$\begin{aligned} \boldsymbol{h}^{(1)}(\boldsymbol{x},t) &= \sigma\big(\boldsymbol{B}(\boldsymbol{x},t)\tilde{\boldsymbol{z}} + \boldsymbol{C}(\boldsymbol{x},t)\gamma(\boldsymbol{x}) + \boldsymbol{\beta}^{(1)}\big), \\ \boldsymbol{h}^{(\ell)}(\boldsymbol{x},t) &= \sigma\big(\boldsymbol{A}^{(\ell)}(\boldsymbol{x},t)\boldsymbol{h}^{(\ell-1)}(\boldsymbol{x},t) + \boldsymbol{\beta}^{(\ell)}\big), \qquad \ell = 2,\dots,L-1, \\ \widetilde{u}(\boldsymbol{x},t) &= \boldsymbol{A}^{(L)}(\boldsymbol{x},t)\boldsymbol{h}^{(L-1)}(\boldsymbol{x},t) + \boldsymbol{\beta}^{(L)}, \end{aligned} \tag{69}$$

where σ is the SiLU activation. The biases $\boldsymbol{\beta}^{(\ell)}$ are shared across patches. The trainable decoder parameters are therefore the hypernetwork parameters ω, the low-rank bases, and the biases.

C.3 Sampled-Query Training

The decoder $h_{\theta_{\mathrm{dec}}}$ in Eq. (12) is implemented as a Neural Field, i.e., a continuous signal representation parameterized by a neural network over spatial coordinates (Sitzmann et al., 2020). This allows the multistep objective in Eq. (3) to be evaluated at arbitrary query locations. During training, we therefore decode only $Q \ll N$ uniformly sampled points at each rollout step rather than the complete spatial discretization. This gives an unbiased estimate of the corresponding full-point loss. Since decoded values never feed back into the latent rollout, query subsampling changes only the supervision cost, not the latent dynamics. The dominant rollout memory decreases from $\mathcal{O}(TN)$ to $\mathcal{O}(T(P+Q))$, while the decoder can still be evaluated densely or at arbitrary coordinates at inference.

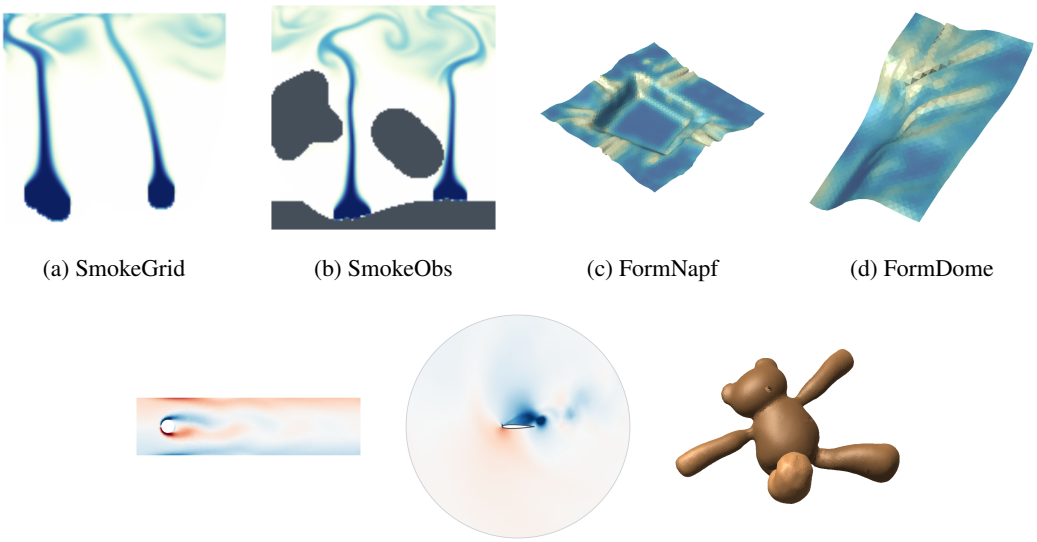
D Datasets

We evaluate seven time-dependent simulation tasks spanning structured grids, irregular meshes, and point clouds. Table 3 summarizes the predicted fields, spatial resolutions, rollout horizons, patch counts, and dataset sizes. The tasks are chosen to cover different forms of localized structure, including thin transported filaments, forming wrinkles, vortex and shock structures, and contact-induced deformation.

Table 3: **Dataset overview.** N denotes the number of observed spatial points, C the number of predicted channels, $N\cdot C$ the resulting number of scalar field values per step, T the rollout horizon, P the number of latent patches, and d_a the dimension of the initial action $\boldsymbol{a}_0$. A dash indicates an unactuated task. The train and test columns give the number of trajectories in each split. Checkpoints are selected using 32 validation trajectories that are separate from the training and test sets.

Task	Geometry	Predicted field	N	C	$N\cdot C$	T	P	train	test	d_a
SmokeGrid	128^2 grid	smoke density	16,384	1	16,384	30	256	3,200	768	N/A
SmokeObs	128^2 grid + obstacle	smoke density	8,003–14,366	1	8,003–14,366	30	256	3,276	788	N/A
FormNapf	lattice, 4 plies	ply positions	3,445	12	41,340	25	323	1,500	300	16
FormDome	lattice, 4 plies	ply positions	1,441	12	17,292	25	135	1,500	300	4
Cylinder	mesh, per trajectory	flow state	$\approx 2{,}000$	3	$\approx 6{,}000$	39	225	800	100	N/A
Airfoil	mesh, graded near wing	flow state	5,233	4	20,932	29	324	800	100	N/A
Teddy	3-D point cloud	point positions	17,323	3	51,969	30	406	800	100	N/A

The two forming tasks predict the positions of four material plies, giving $C = 12$ channels, and are the only actuated problems in the benchmark. The product $N\cdot C$ makes the per-step output size explicit: the forming lattices carry 17,292 and 41,340 values per step despite their small node counts. Cylinder uses a different mesh for every trajectory; the number of patches remains fixed while the spatial cover is reconstructed from each geometry. Airfoil instead uses a single mesh



(a) SmokeGrid (b) SmokeObs (c) FormNapf (d) FormDome

(e) Cylinder (f) Airfoil (g) Teddy

Figure 4: **The seven simulation tasks.** *(a)* SmokeGrid: buoyant smoke forming thin filaments on a regular grid. *(b)* SmokeObs: the same flow around a trajectory-dependent obstacle. *(c)* FormNapf: four composite plies formed into a rectangular cup. *(d)* FormDome: four plies formed over the double-dome geometry. *(e)* Cylinder: incompressible flow past a cylinder on an irregular mesh. *(f)* Airfoil: transonic flow around an airfoil with strong mesh refinement near the wing. *(g)* Teddy: deformation of a soft body under contact with the ground.

shared by all trajectories; only the inflow Mach number and the angle of attack vary between trajectories. SmokeObs varies its geometry by placing a different obstacle in each trajectory, so the number of observed fluid cells ranges from 8,003 to 14,366. For the resolution study in Fig. 3(c), we additionally simulate SmokeGrid at 256^2, 384^2, and 512^2 while keeping $P = 1024$ fixed.

Fig. 4 shows representative states from all seven tasks. The experiments deliberately combine regular and irregular discretizations so that the model is not tied to a particular input geometry.

Buoyant smoke. SmokeGrid models buoyancy-driven smoke emitted continuously from the bottom of a closed domain. The plume rises, folds, and develops thin filaments that must be transported over the rollout. We simulate the flow in NVIDIA Warp (Macklin, 2022) using semi-Lagrangian advection and pressure projection,

$$\partial_t \boldsymbol{v} + (\boldsymbol{v} \cdot \nabla)\boldsymbol{v} = -\nabla p + \beta \rho\, \boldsymbol{e}_{\mathrm{up}}, \qquad \nabla \cdot \boldsymbol{v} = 0, \qquad \partial_t \rho + \boldsymbol{v} \cdot \nabla \rho = s(\boldsymbol{x}),$$

where ρ is the smoke density, β the buoyancy coefficient, $\boldsymbol{e}_{\mathrm{up}}$ the upward unit vector and s the source term. The dataset contains 4,000 trajectories of 40 frames at 128^2: 3,200 are used for training, 32 for validation, and 768 for testing, with $T = 30$ rollout steps.

SmokeObs adds a solid obstacle whose position changes between trajectories, leaving between 8,003 and 14,366 observed fluid cells per trajectory. It contains 3,276 training, 32 validation, and 788 test trajectories. Point-based methods receive the remaining fluid cells directly as a point cloud, while grid Fourier baselines are not evaluated because the obstacle is not represented by an input channel; details are given in Appendix E. For resolution scaling, we regenerate the same physical setup at 256^2, 384^2, and 512^2 and use 1,000 training and 168 test trajectories at each resolution.

Composite forming. FormNapf and FormDome model thermoforming of four fabric plies under frictional tool–ply and ply–ply contact. The gripper configuration is fixed within a trajectory and is the only varying control input, while the press motion is shared across trajectories. FormNapf forms a rectangular napf geometry on a masked 83×83 material lattice with 3,445 nodes and eight grippers, giving a 16-dimensional action (Doehner et al., 2026). FormDome uses the standard double-dome tool, which contains two adjacent domes. The four plies are represented on a masked 43×67 material

lattice with $1{,}441$ nodes and controlled by two grippers, giving a 4-dimensional action (Würth et al., 2025; 2026). Both tasks use $1{,}500$ training and 300 test trajectories with $T = 25$. Ground truth is generated by high-fidelity finite-element simulations with anisotropic, rate-dependent material behavior and frictional contact (Würth et al., 2026; Doehner et al., 2026). The localized quantity of interest is the out-of-plane wrinkling of the plies.

Mesh fluids. Cylinder and Airfoil are taken from the MeshGraphNets datasets (Pfaff et al., 2021). Cylinder uses a separate irregular triangular mesh for each trajectory, while Airfoil uses a single mesh shared by all trajectories. Cylinder contains approximately $2{,}000$ nodes and models incompressible flow past a cylinder whose location and diameter vary between trajectories. Airfoil contains $5{,}233$ nodes and models compressible flow around an airfoil on an unstructured mesh strongly refined near its surface; the training trajectories vary the freestream Mach number from 0.2 to 0.7 and the angle of attack from -25° to 25°. Following IGNS (Hoang et al., 2026), we evaluate the early transient part of the trajectories rather than the full 600 solver steps: Cylinder uses the first 200 steps at stride five with $T = 39$, and Airfoil the first 300 steps at stride ten with $T = 29$. This avoids evaluating primarily on the later near-periodic regime after the wake has settled. Both tasks use 800 training and 100 test trajectories. We predict the flow state on interior and outflow nodes, while wall and inflow nodes are treated as boundary conditions.

Deformable contact. Teddy models a volumetric soft body dropped from a uniformly sampled initial orientation onto a static floor. Ground truth is generated with finite-element soft-body simulation in NVIDIA Isaac (Mittal et al., 2023). The simulation volume is discretized by $33{,}274$ tetrahedra with $17{,}323$ vertices. The $20{,}376$ boundary triangles reference $10{,}157$ vertices, leaving $7{,}166$ interior vertices that are predicted together with the surface. All methods receive the vertex positions as an unstructured point cloud without tetrahedral connectivity. The dataset contains $1{,}000$ trajectories of 32 frames, with 100 test trajectories and a rollout horizon of $T = 30$. The setting follows the deformable-contact benchmark of Dahlinger et al. (2026), but at substantially higher spatial resolution. Deformation is concentrated around the contact region, making Teddy a point-cloud test of localized curvature preservation.

E BASELINES

We compare against methods that differ primarily in where they perform the temporal rollout. Latent simulators encode the physical field once and evolve a compressed state through time,

$$\boldsymbol{z}_{t_0} = \mathrm{Encoder}(u(\cdot, t_0)), \qquad \boldsymbol{z}_{t+1} = \mathrm{Processor}(\boldsymbol{z}_t), \qquad \widehat{u}(\boldsymbol{x}, t+1) = \mathrm{Decoder}(\boldsymbol{x}, \boldsymbol{z}_{t+1}),$$

whereas one-step operators reconstruct the physical field after every update and use it as input to the next step,

$$\widehat{u}(\boldsymbol{x}, t+1) = \mathrm{Decoder}(\mathrm{Processor}(\mathrm{Encoder}(\widehat{u}(\cdot, t)))) \,.$$

Here $\boldsymbol{x}$ denotes the spatial coordinates. The comparison separates three properties: the latent-state representation, whether this state persists across steps or is reconstructed from the physical field, and the temporal supervision used for training. The equations above distinguish the two rollout mechanisms. Table 4 compares the methods along these axes, while Table 5 summarizes their architectural components. Unless stated otherwise, all baselines follow the same wall-clock protocol as SPANS.

CALM-PDE. CALM-PDE (Hagnberger et al., 2025) is the closest latent baseline. It maps the input field to a fixed set of learned spatial query points using local continuous convolutions, evolves the resulting latent states through time, and continuously decodes them to arbitrary output coordinates. The processor is formulated as a neural ODE and implemented as a single explicit Euler step of unit length, with the displacement computed by a distance-biased Transformer attention. The query locations are optimized across the training set and remain fixed after training, whereas SPANS obtains its coarse patch centers directly from the reference geometry, without training them. We increase the number of CALM-PDE latent tokens from 16 to 64 to strengthen the comparison. Larger token sets exceed the 40 GB GPU memory budget. On the second-order tasks, we follow the default setting in the released code where CALM-PDE jointly encodes the preceding and current position fields. It maintains no explicit velocity state, so the initial velocity enters only through this encoded history.

Table 4: **Comparison along architectural and training axes.** *Latent rollout* denotes encoding the field once and evolving the resulting representation instead of reconstructing and re-encoding the physical field at each step. *Reduced* indicates that the latent representation is smaller than the input discretization. *Fixed structure* indicates a spatial layout prescribed by the geometry or a grid rather than learned tokens or query points. *Multistep training* supervises the entire model's rollout instead of one transition. † marks methods whose released recipe trains single-step; their multistep training is our adaptation. SPANS combines all four properties. Group labels follow Table 1.

	Method	Latent rollout	Reduced	Fixed structure	Multistep training
(I)	FNO	×	×	N/A	✓
	LOGLO-FNO	×	×	N/A	✓†
(II)	GAOT	×	✓	✓	×
	Transolver	×	✓	×	×
	Transolver-T	×	✓	×	✓†
(III)	UPT	✓	✓	×	✓†
	CALM-PDE	✓	✓	×	✓
	AROMA	✓	✓	×	×
(IV)	IGNS	✓	×	N/A	✓
	MGN	×	×	N/A	×
	SPANS (ours)	✓	✓	✓	✓

Table 5: **Architectural decomposition of the compared methods.** The encoder maps the observed field to the latent representation, the processor evolves this representation, and the decoder evaluates the predicted field at query positions. A dash denotes a method without a reduced representation. The latent grid and patch graph have spatial layouts prescribed by the grid or geometry, while feature slices, token sets, and point sets are learned: tokens carry no explicit spatial location, whereas each point-set state is tied to a learned spatial query location. Transolver-T shares Transolver's architecture (Table 4). The methods are described individually below.

	Method	Encoder	Representation	Processor	Decoder
(I)	FNO	linear	N/A	global spectral	MLP
	LOGLO-FNO	linear	N/A	local + global spectral	MLP
(II)	GAOT	graph pooling	latent grid	transformer	graph query
	Transolver	MLP	feature slices	slice attention	linear
(III)	UPT	attention	token set	transformer	attention
	CALM-PDE	cont. conv	point set	transformer	cont. conv
	AROMA	attention	token set	diffusion transformer	neural field
(IV)	IGNS	linear	N/A	Hamiltonian MP	linear
	MGN	MLP	N/A	message passing	MLP
	SPANS (ours)	local pooling	patch states	short- + long-range attn.	POU neural field

UPT. UPT (Alkin et al., 2024) pools the input points onto random supernodes by message passing and then into a fixed number of latent tokens by Perceiver-style cross-attention. A Transformer is applied to evolve thoes latent tokens, and finally decoded back to the output coordinates via cross-attention from the query coordinates. The tokens have no fixed spatial layout, unlike the patch states of SPANS. We run UPT at the authors' published two-million-parameter configuration (512 latent tokens, supernodes 12.5% of the input points). For the processor module, the fluid tasks use the first-order Euler integration while in the deformable tasks we ran both the semi-implicit Euler update of Eq. (60), and the released first-order integration with two input frames. The second-order integration is less stable on FormNapf and FormDome and diverges there. Meanwhile, this integration is more accurate on Teddy, probably due to the elastic properties of the material. In the main results, we therefore report UPT with the semi-implicit update on Teddy and the two-frame variant on FormNapf and FormDome.

Transolver. Transolver (Wu et al., 2024) fixes the number M of physics-aware tokens but not their spatial support. In each Physics-Attention block, a learned projection softly assigns every point to M latent slices, and each token is formed by a normalized weighted average of the corresponding point features. Self-attention updates these tokens, which are then mapped back to the points using the same assignment weights. Because the assignments and token values are recomputed from the current features, they have no fixed spatial layout and are not carried between rollout steps. We evaluate the authors' default one-step training, denoted *Transolver*, and the same architecture trained with our multistep rollout objective, denoted *Transolver-T*. On the second-order tasks, Transolver predicts nodal acceleration and follows the semi-implicit Euler update in Eq. (60).

GAOT. GAOT (Wen et al., 2025) encodes the input points onto a regular latent grid with a multiscale graph neural operator over a fixed radius, applies a Transformer to the patchified grid, and decodes back to arbitrary coordinates with the same operator. The grid has a fixed spatial layout but is not evolved: one call maps the field at t to $t + \tau$, and the rollout re-encodes the predicted field at every step. We use the released encoder, decoder, processor, and geometry embeddings. Because its fine-to-latent aggregation uses a fixed spatial radius, the number of observations associated with a token can vary substantially on strongly graded meshes such as Airfoil. For Teddy, we use a k-nearest-neighbor rule instead of the released radius rule. All other tasks retain the released radius rule. We train GAOT with its released protocol, where it maps a single input frame to the state a lead time τ ahead, conditioned on the start and lead time, and is trained on single input-output pairs at variable τ, i.e.,

$$\mathcal{L}_{\mathrm{GAOT}}(\theta) = \mathbb{E}_{u,t,\tau} \left\| u(t+\tau) - \widehat{\mathcal{G}}_\theta(u(t), t; \tau) \right\|_2 . \tag{70}$$

Graph simulators. We additionally compare against IGNS (Hoang et al., 2026) and MeshGraphNets (MGN) (Pfaff et al., 2021) on tasks where the underlying mesh representation is available. IGNS projects the physical nodes to hidden states once and evolves them with one message-passing layer per rollout step, giving a latent rollout whose state still contains one feature vector per input node. Its port-Hamiltonian update combines the gradient of a learned energy with a separate external-forcing term. MGN instead applies 15 rounds of message passing to the current fine-resolution graph and predicts the next physical state, following the original one-step teacher-forced training procedure. Both therefore retain spatial states whose size grows with the input discretization, in contrast to the fixed-size latent representations used by SPANS, CALM-PDE, and UPT.

AROMA. AROMA (Serrano et al., 2024) is trained in two stages. The first stage jointly trains an attention-based encoder and a continuous local neural-field decoder as a variational autoencoder. The encoder compresses the observed field into a fixed-size sequence of latent tokens whose spatial roles are learned through geometry-dependent attention rather than tied to a prescribed patch cover. The decoder reconstructs the field at arbitrary query coordinates. The second stage freezes the encoder and decoder and trains a conditional diffusion Transformer on successive latent states. Following the released implementation, the Transformer learns a one-step transition by denoising the next latent state conditioned on the current one. At evaluation, this transition is applied autoregressively in latent space, and the reported training cost includes both stages.

FNO and LOGLO-FNO. FNO (Li et al., 2021) and LOGLO-FNO (Kalimuthu et al., 2025) serve as spectral grid references. Both evolve the complete state on a dense rectangular lattice. FNO applies truncated Fourier convolution over the full lattice. LOGLO-FNO shares our goal of capturing localized fine-scale structure: it adds a parallel branch that retains all Fourier modes within small non-overlapping patches, modeling this structure locally without retaining the corresponding modes globally. Unlike the overlapping cover of SPANS, these patches are disjoint and impose no explicit continuity constraint at their interfaces. SmokeGrid directly provides a dense lattice, although its physical boundaries are non-periodic. For the forming tasks, we scatter the material state to a padded rectangular grid and provide the occupancy mask as an input channel. Because the authors' code is unavailable, our reimplementation contains the local–global Fourier blocks but not the separate high-frequency propagation module or frequency-aware loss described in the paper. On the forming tasks, both networks predict nodal acceleration and follow the semi-implicit Euler update in Eq. (60).

Action conditioning. The forming tasks provide only one initial gripper action $\boldsymbol{a}_0$ to every method. FNO, LOGLO-FNO, Transolver, and UPT concatenate this fixed input to the physical-

Table 6: **Per-task SPANS settings.** P is the number of patches, J the latent width, and Q the number of decoded supervision points per step; "full" denotes decoding all points. Order 1 uses the first-order latent update and order 2 the second-order formulation of Appendix C.1.

Task	lr ($\times 10^{-4}$)	batch	P	J	Q	Order
SmokeGrid	3.0	8	256	64	2048	1
SmokeObs	3.0	2	256	64	2048	1
Cylinder	5.0	8	225	24	full	1
Airfoil	1.5	8	324	64	1024	1
FormDome	6.0	16	135	128	full	2
FormNapf	4.5	16	323	128	512	2
Teddy	4.5	8	406	128	1024	2

Table 7: **Wall-clock training budgets.** Nominal training budget for each task and runs that deviate from it. All experiments use one NVIDIA A100.

Task	Budget (h)	Deviations
SmokeGrid	24	AROMA 34 h
SmokeObs	24	AROMA 34 h
FormNapf	48	none
FormDome	24	none
Cylinder	12	IGNS, MGN 24 h
Airfoil	24	CALM-PDE $\sim$ 40 h; AROMA 32 h
Teddy	24	Transolver-T $\sim$ 40 h
Resolution 256^2	48	none
Resolution 384^2, 512^2	72	none

point features whenever the physical field is processed. CALM-PDE and SPANS instead inject its embedding once into the latent state at the beginning of the rollout.

F Hyperparameters and training protocol

All SPANS experiments use AdamW with weight decay 10^{-4}, gradient clipping at 1.0, and a constant learning rate. We optimize SPANS using the relative L_2 error over autoregressive rollout windows, with a linear curriculum on the rollout horizon and Gaussian input noise with standard deviation 0.01. The baselines retain the training objectives of their released implementations, i.e., either relative L_2 error or MSE on normalized features. Unless stated otherwise, the architecture is fixed across tasks: a DeepSets encoder, two processor blocks, each combining short-range and long-range attention with equal branch weights, and a rank-128 generated decoder with generator width 256. The support radius ρ of the Wendland weights in Eq. (17) is set per patch, $\rho_p = 2.5\, h_p$, where h_p denotes the local patch spacing; the partition uses a support-complete stencil.

Task-specific choices are limited to the learning rate, batch size, number of patches P, latent width J, number of decoded supervision points Q, and update order; these choices are optimal values made by running a small grid search and are listed in Table 6. In addition, we generate patches by tiling the domain into regular rectangles for Cylinder and Airfoil, while for the point-cloud tasks FormNapf, FormDome, and Teddy, we use k-means clustering to obtain patch centers. The deformable tasks (FormNapf, FormDome, and Teddy) use the second-order protocol in Appendix C.1; all remaining tasks are first order. Checkpoints are selected by validation error. The resolution-scaling experiments reuse the SmokeGrid configuration with $P = 1024$.

Compute budget. All experiments are trained on a single NVIDIA A100 with 40 GB of memory. We use a fixed wall-clock budget for each task, reported in Table 7, and allocate the same or greater training time to the compared baselines whenever their training pipeline permits. The table records all runs that deviate from the nominal budget. For multi-stage methods such as AROMA, the reported training time is the sum over all stages.

Baseline configurations. We use the released architectures and configurations of each baseline where possible. The detailed implementation of each baseline and the reasons for individual changes are given in Appendix E. All methods are evaluated over the same task-specific rollout horizons used by SPANS.

G METRIC DEFINITIONS

All methods are evaluated by free rollout from the initial frames using the selected validation checkpoint. The primary prediction error is the *rollout-averaged MSE*: the error averaged over every step of the free rollout, not the error at its last step. The local-detail metrics below are instead evaluated at the final rollout frame.

Position and field error. For the forming and Teddy tasks, position MSE is evaluated after undoing the data normalization and averaged over rollout steps, trajectories, spatial points, and position channels. For the smoke tasks, we analogously report MSE of the predicted density over the complete rollout; on SmokeObs, obstacle cells are zero-masked before scoring. For Cylinder and Airfoil, the main tables report MSE of the normalized flow state over the predicted interior and outflow nodes and all state channels. Wall and inflow nodes are prescribed boundary conditions and are excluded.

For the appendix results on the fluid tasks, we additionally report relative L_2 error in physical state space. For each rollout step and channel, it is computed over the predicted nodes as

$$e_{\mathrm{rel}} = \frac{\|\widehat{u} - u\|_2}{\|u\|_2}, \tag{71}$$

and then averaged over rollout steps, channels, and trajectories.

Crease structure. For the forming tasks, position error can obscure narrow wrinkles. Following common practice in composite forming, we therefore evaluate their out-of-plane amplitude and spatial placement (Arnold et al., 2016; Boisse et al., 2021; Shen et al., 2023). For every material node with a complete stencil, we isolate this local variation and compute

$$(Dz)_i = 4z_i - \sum_{j \in \mathcal{N}_4(i)} z_j, \qquad a_z = \frac{\|D\widehat{z}\|_2}{\|Dz\|_2}, \qquad \rho_z = \frac{\langle D\widehat{z}, Dz\rangle}{\|D\widehat{z}\|_2 \|Dz\|_2}. \tag{72}$$

Here $\mathcal{N}_4(i)$ contains the four nearest one-ring neighbors, and z contains the out-of-plane coordinates of all four plies at the final rollout step. The amplitude ratio a_z measures crease strength, while the cosine similarity ρ_z measures whether the creases occur at the correct locations. Both are best near one.

Point-cloud curvature. For Teddy, we use local curvature to measure whether contact-induced deformation is preserved. We construct a fixed 6-nearest-neighbor graph from the rest-state point cloud and compute

$$(Lx)_i = x_i - \frac{1}{6} \sum_{j \in \mathcal{N}_6(i)} x_j, \qquad a_{\mathrm{curv}} = \frac{\|L\widehat{x}\|_2}{\|Lx\|_2}, \qquad \rho_{\mathrm{curv}} = \frac{\langle \widetilde{L\widehat{x}}, \widetilde{Lx}\rangle}{\|\widetilde{L\widehat{x}}\|_2 \|\widetilde{Lx}\|_2}. \tag{73}$$

Here $\widetilde{f} = f - \overline{f}$ denotes mean-centering. The amplitude ratio a_{curv} measures the amount of local curvature, while the cosine similarity ρ_{curv} measures its spatial placement. The similarity is computed per trajectory and then averaged over the test set; both metrics are best near one.

High-frequency spectral magnitude. For the smoke tasks, we measure whether the final prediction retains the same amount of fine-scale variation as the reference. We compute the two-dimensional discrete Fourier transform and select components whose combined spatial frequency exceeds half the maximum frequency represented by the grid. This band captures patterns that vary over fewer than four grid spacings. Following the banded Fourier evaluation of

PDEBench (Takamoto et al., 2022), let $\mathcal{B}_{\mathrm{HF}}$ denote these components. We compute their mean magnitude and the predicted-to-reference ratio

$$A_{\mathrm{HF}}(u) = \frac{1}{|\mathcal{B}_{\mathrm{HF}}|} \sum_{q \in \mathcal{B}_{\mathrm{HF}}} |\mathcal{F}_{\mathrm{2D}}(u)_q|, \qquad r_{\mathrm{HF}} = \frac{A_{\mathrm{HF}}(\widehat{u}_T)}{A_{\mathrm{HF}}(u_T)}. \tag{74}$$

Here T is the final rollout step. A ratio near one indicates that the prediction contains the correct amount of fine-scale variation. Values below one indicate excessive smoothing, while values above one indicate excess fine-scale variation, which may include noise.

For SmokeObs, the obstacle cells contain no solution and must be filled before applying the Fourier transform. Setting them to zero creates a sharp discontinuity at the obstacle boundary, introducing artificial high-frequency content unrelated to the fluid. We therefore complete the prediction and reference separately using a discrete harmonic extension. Let $\mathcal{F}$ and $\mathcal{W}$ denote the fluid and obstacle cells, respectively. For either field v, its extension $\widetilde{v}$ is defined by

$$\widetilde{v}_i = \begin{cases} v_i, & i \in \mathcal{F}, \\ \dfrac{1}{4} \displaystyle\sum_{j \in \mathcal{N}_4(i)} \widetilde{v}_j, & i \in \mathcal{W}, \end{cases} \tag{75}$$

where $\mathcal{N}_4(i)$ contains the four periodic grid neighbors. This extension leaves the fluid unchanged and smoothly continues its values through the obstacle, removing the artificial high-frequency content introduced by zero filling. Consequently, r_{HF} measures variation in the predicted fluid rather than the fixed obstacle boundary. A spatially constant prediction therefore attains $r_{\mathrm{HF}} = 0.00$, compared with 0.56 under zero filling. For SmokeGrid, which has no obstacle cells, the definition is unchanged.

H Additional results

In addition to the main-text results, we include more extensive comparisons on local-detail metrics and on data efficiency. We also provide further ablation experiments and a detailed comparison of our results with those reported in the original papers for the baseline methods.

H.1 Complete fluid-task results

Table 8 reports the complete results on the four fluid tasks, including the whole-mesh graph baselines IGNS and MGN omitted from the main comparison for compactness. In addition to rollout MSE, we report relative L_2 error in physical state space. Both metrics give the same overall conclusion: SPANS performs best on Airfoil, SmokeGrid, and SmokeObs, while ranking third on Cylinder behind Transolver and CALM-PDE.

The advantage is strongest on Airfoil, where SPANS reduces MSE by $3.0\times$ relative to the next-best method, IGNS, and on the two smoke tasks, where it also achieves the lowest error. Cylinder is the only fluid task on which SPANS does not lead, with Transolver and CALM-PDE performing better. The qualitative rollout in Fig. 14 helps explain this result. SPANS recovers and maintains the characteristic periodic vortex street, but gradually accumulates phase error relative to the reference. In contrast, AROMA extrapolates the dynamics but increasingly places the wake structures at incorrect spatial locations. Transolver and CALM-PDE track the periodic shedding more accurately on this task. Thus, the higher error of SPANS on Cylinder reflects inaccurate phase propagation rather than a failure of its compressed representation to recover the underlying flow pattern.

H.2 Local-detail results

The main rollout metric alone cannot distinguish an accurate prediction from one that reduces error by smoothing localized structure. Table 9 therefore complements rollout MSE with task-specific measures of high-frequency magnitude, local curvature, and wrinkle placement; their definitions are given in Appendix G. Across all three task families, SPANS combines low rollout error with strong preservation of the corresponding local structures.

Table 8: **Complete fluid-task results.** Rollout-averaged field MSE ($\times 10^{-2}$) and relative L_2 error ($\times 10^{-1}$). Lower is better for both metrics. Groups follow Table 1; group *(IV)* contains whole-mesh graph rollouts. Best and second-best (average) results are highlighted in **orange** and **teal**, respectively.

		Cylinder		Airfoil	
	Method	MSE	rel. L_2	MSE	rel. L_2
(II)	GAOT	36.0±4.2	6.370±0.437	35.7±16.0	1.010±0.077
	Transolver	1.13±0.026	0.656±0.009	4.85±0.54	0.416±0.008
	Transolver-T	3.64±0.09	1.140±0.027	22.4±0.15	1.330±0.016
(III)	UPT	7.84±1.1	2.010±0.200	12.3±1.1	1.080±0.039
	CALM-PDE	1.64±0.13	0.837±0.057	5.55±1.1	0.637±0.070
	AROMA	10.3±0.58	1.530±0.047	8.68±1.1	0.482±0.039
(IV)	IGNS	7.32±0.58	1.860±0.079	4.83±0.83	0.621±0.043
	MGN	58.0±13	3.220±0.784	58.3±7.6	1.930±0.312
	SPANS (ours)	2.40±0.26	1.040±0.016	1.59±0.095	0.249±0.017
		SmokeGrid		**SmokeObs**	
	Method	MSE	rel. L_2	MSE	rel. L_2
(I)	FNO	0.933±0.066	2.520±0.135	N/A	N/A
	LOGLO-FNO	0.803±0.021	2.260±0.037	N/A	N/A
(II)	GAOT	0.368±0.007	1.550±0.018	0.545±0.008	2.090±0.011
	Transolver	1.370±0.046	3.090±0.060	1.500±0.042	3.630±0.043
	Transolver-T	2.180±0.535	4.630±0.634	1.660±0.088	4.280±0.108
(III)	UPT	3.580±2.600	6.030±2.400	3.450±1.600	6.320±1.600
	CALM-PDE	0.424±0.016	1.820±0.032	0.785±0.035	2.780±0.064
	AROMA	1.150±0.025	3.020±0.033	1.510±0.032	3.810±0.040
(IV)	IGNS	2.090±0.034	4.650±0.041	1.910±0.065	4.660±0.067
	MGN	6.690±2.250	7.780±1.360	3.950±0.852	6.230±0.536
	SPANS (ours)	0.231±0.011	1.230±0.021	0.443±0.012	1.990±0.032

Smoke. On SmokeGrid and SmokeObs, SPANS achieves the lowest rollout MSE while retaining high-frequency magnitude ratios of 0.952 and 0.772, respectively. The baselines reveal why spectral magnitude must be considered together with prediction accuracy. FNO and LOGLO-FNO reproduce the overall high-frequency magnitude particularly closely on SmokeGrid, but with substantially larger rollout error, whereas GAOT systematically introduces excess high-frequency content and CALM-PDE and UPT attenuate it. On SmokeObs, both Transolver variants attain ratios closest to one (1.035 and 1.112) but incur three to four times the rollout error ofSPANS. These results together illustrate that matching the overall high-frequency magnitude alone does not ensure an accurate fine-scale reconstruction. The qualitative rollout in Fig. 16 shows the corresponding spatial behavior: SPANS preserves the transported filament structure without either suppressing it or introducing the excess fine-scale variation observed in competing predictions. Thus, the improvement is not simply due to retaining more high-frequency energy, but to retaining the fine-scale structure at the correct locations.

Teddy. A similar distinction appears on the unstructured Teddy point cloud. SPANS obtains the lowest position MSE, 0.633, with a curvature correlation of 0.941 and a curvature amplitude of 0.954. UPT reaches a slightly higher curvature correlation, 0.959, at a similar amplitude, but with an about $4\times$ larger position error, and Transolver-T reproduces the overall curvature magnitude slightly more closely, at 0.990, with a larger position error still. Hence, matching the amount of local deformation is again insufficient if that deformation occurs at the wrong spatial locations. The qualitative rollout in Fig. 17 confirms that SPANS more accurately preserves the localized deformation through contact and subsequent settling.

Composite forming. The forming tasks provide the strongest test of spatial localization because the relevant structures are narrow wrinkles superimposed on a large global deformation. On both FormNapf and FormDome, SPANS achieves the lowest position error and the highest crease-placement correlation ρ_z. Its crease amplitude is not always closest to one: GAOT and LOGLO-

Table 9: **Local-detail results.** Rollout error and task-specific local-detail metrics for the smoke tasks (top), Teddy (middle), and the forming tasks (bottom). MSE is reported as density MSE ($\times 10^{-2}$) for the smoke tasks, position MSE ($\times 10^{-3}$) for Teddy, and unscaled position MSE for the forming tasks. A dash indicates that the method is not applicable. Best and second-best (average) MSE are highlighted in **orange** and **teal**, respectively.

		SmokeGrid		SmokeObs	
	Method	MSE ↓	high-freq.	MSE ↓	high-freq.
(I)	FNO	0.933±0.066	1.010±0.016	N/A	N/A
	LOGLO-FNO	0.803±0.021	1.000±0.006	N/A	N/A
(II)	GAOT	0.368±0.007	1.320±0.006	0.545±0.008	1.527±0.037
	Transolver	1.370±0.046	1.200±0.255	1.500±0.042	1.035±0.052
	Transolver-T	2.180±0.535	1.090±0.058	1.660±0.088	1.112±0.018
(III)	UPT	3.580±2.600	0.139±0.099	3.450±1.600	0.083±0.058
	CALM-PDE	0.424±0.016	0.729±0.040	0.785±0.035	0.496±0.011
(IV)	IGNS	2.090±0.034	1.020±0.080	1.910±0.065	1.251±0.340
	SPANS (ours)	0.231±0.011	0.952±0.006	0.443±0.012	0.772±0.046

		Teddy		
	Method	Position MSE ↓	Curvature corr. ↑	Curvature amp.
(II)	GAOT	4.600±2.190	0.751±0.037	1.290±0.042
	Transolver	246.0±32.9	0.401±0.015	1.184±0.051
	Transolver-T	5.350±1.310	0.930±0.009	0.990±0.003
(III)	UPT	2.500±0.360	0.959±0.003	0.953±0.015
	CALM-PDE	1.970±0.055	0.798±0.008	1.120±0.009
	SPANS (ours)	0.633±0.015	0.941±0.007	0.954±0.008

		FormNapf			FormDome		
	Method	MSE ↓	ρ_z	a_z	MSE ↓	ρ_z	a_z
(I)	FNO	1.020±0.007	0.421±0.006	0.701±0.004	1.010±0.052	0.328±0.027	0.594±0.063
	LOGLO-FNO	1.030±0.013	0.406±0.003	0.836±0.010	0.951±0.031	0.362±0.003	0.780±0.023
(II)	GAOT	0.900±0.019	0.374±0.006	0.848±0.016	0.941±0.036	0.328±0.005	0.728±0.016
	Transolver	89.710±2.480	0.056±0.042	1.200±0.682	34.390±0.676	0.124±0.060	0.926±0.306
	Transolver-T	1.070±0.013	0.446±0.002	0.503±0.015	1.020±0.043	0.300±0.021	0.486±0.026
(III)	UPT	1.430±0.031	0.423±0.006	0.492±0.022	1.050±0.081	0.380±0.003	0.421±0.012
	CALM-PDE	0.876±0.015	0.406±0.007	0.616±0.005	0.951±0.042	0.376±0.014	0.477±0.007
	SPANS (ours)	0.799±0.016	0.448±0.002	0.613±0.005	0.922±0.044	0.403±0.008	0.463±0.018

FNO, for example, retain stronger high-frequency amplitudes on FormNapf. However, Fig. 13 shows that much of this additional response appears as poorly localized wrinkle structure or noise. In contrast, SPANS produces lower overall error while placing the predicted wrinkles more consistently with the reference. The one-step Transolver objective fails particularly strongly on these action-conditioned rollouts, while multi-step Transolver-T remains stable but produces substantially weaker crease amplitudes.

Together, these results sharpen the observation from the main comparison. The benefit of the structured patch representation is not merely that it preserves high-frequency content under compression; rather, it retains localized variation at the spatial locations where it occurs. This distinction is consistent across regular grids, structured forming meshes, and an unstructured three-dimensional point cloud.

H.3 Scaling details

We provide the complete results behind the scaling studies in Figs. 3 and 6. They examine three distinct questions: whether the advantage persists with less training data, how rollout stability changes

Table 10: Data-efficiency on the smoke tasks. Rollout-averaged MSE ($\times 10^{-2}$) for different numbers of training trajectories. The final row uses the complete training set of 3,200 trajectories for SmokeGrid and 3,276 for SmokeObs. Lower is better.

	SmokeGrid			SmokeObs		
n_{train}	SPANS	GAOT	CALM-PDE	SPANS	GAOT	CALM-PDE
100	1.193	1.215	1.789	1.557	1.520	1.854
250	0.732	0.794	1.194	1.113	1.195	1.379
500	0.506	0.590	0.840	0.842	0.876	1.085
1000	0.371	0.470	0.630	0.609	0.653	0.869
2000	0.273	0.399	0.486	0.469	0.572	0.762
full	0.244	0.376	0.441	0.451	0.536	0.774

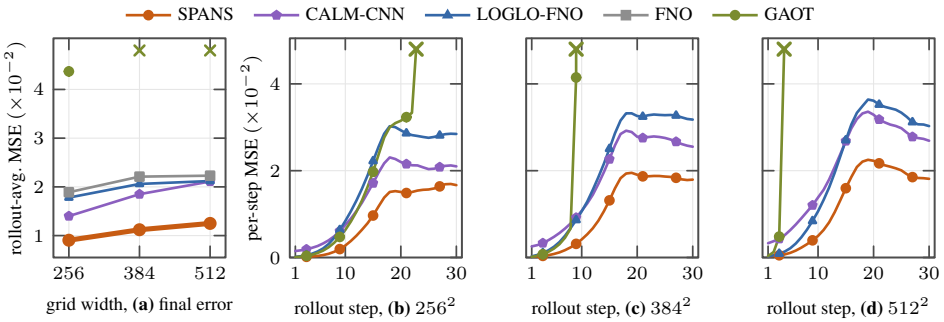


Figure 5: **Resolution ladder on SmokeGrid.** *(a)* Rollout-averaged MSE at each resolution, additionally including FNO and CALM-CNN (the grid CNN variant of CALM-PDE). *(b–d)* Per-step rollout MSE at 256^2, 384^2, and 512^2. Crosses mark where a method exceeds the plotted range.

with observation resolution, and whether additional latent capacity is better spent on more spatial patches or wider patch states.

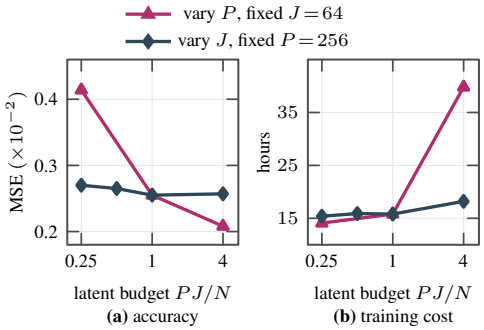
Data efficiency. Table 10 shows that the advantage of SPANS is already present with limited training data and persists as the dataset grows. On SmokeGrid, SPANS achieves the lowest error at every dataset size; with only 100 trajectories, its MSE of 1.193 already matches CALM-PDE trained with 250 trajectories (1.194). On SmokeObs, GAOT is slightly better at 100 trajectories, but SPANS becomes best from 250 trajectories onward. With the complete datasets, SPANS reaches MSEs of 0.244 and 0.451, compared with 0.376 and 0.536 for GAOT and 0.441 and 0.774 for CALM-PDE. The improvements in the main comparison therefore do not depend on the large-data regime.

Resolution and error accumulation. Fig. 5(a) completes the resolution comparison of Fig. 3(c) with the additional grid-only baselines, including FNO and CALM-CNN (the grid CNN variant of CALM-PDE). SPANS achieves the lowest MSE at every resolution: 0.905, 1.118, 1.254, followed by CALM-CNN, the strongest baseline with MSEs of 1.396, 1.846, 2.114 at $256^2, 384^2, 512^2$, respectively. Fig. 5 shows that GAOT exceeds the plotted range near steps 23, 9, and 4 at 256^2, 384^2, and 512^2, respectively, while SPANS remains bounded. During training, GAOT predicts single next states and uses autoregressive rollout for inference. It must therefore decode and re-encode every prediction, potentially encountering states never seen during training. This mismatch between training and inference regimes may explain the faster error accumulation at higher resolutions.

Allocating latent capacity. Finally, we examine how model capacity should be allocated within the latent representation. Fig. 6(a) shows that increasing the number of patches P is substantially more effective than increasing the latent width J. The largest gain occurs when moving from a coarse patch representation to $P = 256$, while further increasing P to 1024 yields more modest improvements. In contrast, increasing the latent width at fixed $P = 256$ has little effect once each patch has sufficient representational capacity. Fig. 6(b) shows the corresponding computational trade-off: increasing P from 256 to 1024 raises the training time from 15.8 to 39.8 hours, whereas widening J from 16 to 256 raises it only from 15.4 to 18.2 hours.

Table 11: **Patch-count and latent-width study on SmokeGrid.** All settings are trained for 150k steps with batch size 8. PJ/N measures the latent-state size relative to the number of input points. The MSE ratio is relative to the default $P = 256$, $J = 64$ setting, denoted by $1.00\times$. Training time is reported in hours.

P	J	PJ/N	MSE	MSE ratio	hours
16	1024	1	0.632	2.48×	26.8
32	64	1/8	0.549	2.15×	13.4
64	64	1/4	0.414	1.62×	14.1
64	256	1	0.388	1.52×	15.8
128	64	1/2	0.367	1.44×	13.9
256	16	1/4	0.270	1.06×	15.4
256	32	1/2	0.265	1.04×	15.9
256	64	1	0.255	1.00×	15.8
256	256	4	0.257	1.01×	18.2
512	32	1	0.269	1.06×	21.2
512	64	2	0.248	0.97×	21.6
1024	16	1	0.226	0.89×	38.7
1024	32	2	0.210	0.82×	38.9
1024	64	4	0.208	0.82×	39.8

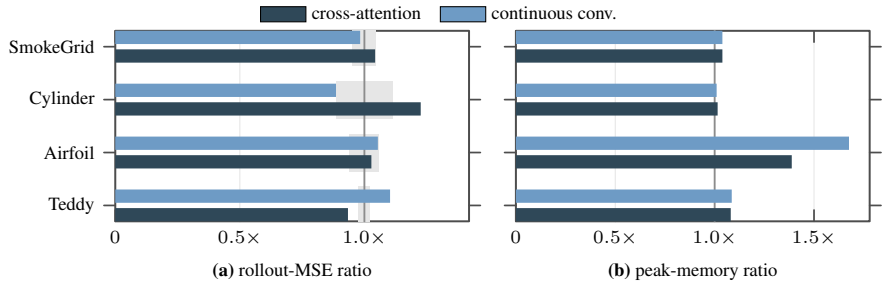


(a) accuracy **(b)** training cost

Figure 6: **Latent-capacity scaling on SmokeGrid.** *(a)* Rollout-averaged MSE against the normalized latent budget PJ/N, varying either the number of patches P at fixed $J = 64$ or the latent width J at fixed $P = 256$, and *(b)* the corresponding training cost. Lower is better in both.

Table 11 gives the complete results behind these trends. At the same overall latent budget, increasing the cover from $P = 16$ to $P = 256$ reduces the MSE to $0.40\times$ its value at $P = 16$, showing that distributing capacity across space is particularly important when the latent cover is coarse. Beyond this point, further refinement yields smaller gains: $P = 1024$ reaches $0.82\times$ the MSE of $P = 256$ at the same width. Likewise, at $P = 256$, increasing the latent width from $J = 16$ to $J = 256$ reduces the MSE only to $0.95\times$ its value at $J = 16$. This suggests that the dominant bottleneck is the spatial resolution of the latent representation rather than the number of features stored at each patch.

The same trend motivates our default choice of $P = 256$, corresponding to a 16×16 cover on SmokeGrid. At this resolution, most of the benefit from spatial refinement has already been realized, while substantially finer covers provide only moderate additional accuracy at a much higher computational cost. We therefore use $P = 256$ as a practical balance between spatial expressiveness and efficiency. More generally, this study suggests that latent capacity is best increased by refining the spatial cover until this regime is reached, rather than by making already sufficiently expressive patch states wider.



(a) rollout-MSE ratio **(b)** peak-memory ratio

Figure 7: **Encoder aggregation ablation.** Budget-matched comparison of DeepSets, cross-attention, and continuous convolution, each read at its own best checkpoint. Panels report *(a)* rollout-MSE and *(b)* peak-memory ratios relative to DeepSets, with $1.0\times$ denoting no change. Lower ratios are better in both panels. Grey bands in *(a)* show the standard deviation of the full model from the main comparison. Memory ratios should be compared only within each task.

Table 12: **Short-range and long-range coupling.** Rollout MSE for the full processor and variants trained after removing one attention branch. Parentheses report the ratio to the full model, where full model values are taken from the main comparison.

Task	Full model	Long-range only	Short-range only
SmokeGrid ($\times 10^{-2}$)	0.231 ± 0.011	0.272 $(1.18\times)$	0.230 $(0.99\times)$
FormNapf	0.799 ± 0.016	0.811 $(1.02\times)$	0.847 $(1.06\times)$
FormDome	0.922 ± 0.044	0.855 $(0.93\times)$	0.931 $(1.01\times)$
Cylinder ($\times 10^{-2}$)	2.40 ± 0.26	1.97 $(0.82\times)$	3.98 $(1.66\times)$
Airfoil ($\times 10^{-2}$)	1.59 ± 0.095	1.49 $(0.94\times)$	2.18 $(1.37\times)$
Teddy ($\times 10^{-3}$)	0.633 ± 0.015	0.608 $(0.96\times)$	1.298 $(2.05\times)$

H.4 Ablation study

Encoder architecture. Fig. 7 compares three different point set aggregation strategies in the encoder, including DeepSet (a shared pointwise MLP followed by mean and max pooling), cross-attention (which learns content- and position-dependent weights), and continuous convolution. Overall, all three encoders achieve comparable MSE values, highlighting that the latent representation itself, rather than the encoder architecture, is the primary factor determining performance. This motivates our design choice to use DeepSets, which is simpler and uses less memory than the other two encoders. The memory increase of the encoder is partially dependent on the density of points in the patches. On Airfoil, where points are dense near the wing, cross-attention and continuous convolution require significantly more memory than DeepSets, while the memory increase is smallest on FormNapf, where points are more evenly distributed.

Short-range and long-range coupling. Table 12 compares the full processor with variants trained after removing one attention branch. Removing long-range attention raises MSE to a significantly higher value, with a $1.66\times$ increase on Cylinder and $2.05\times$ on Teddy. Meanwhile, removing short-range attention degrades performance on SmokeGrid. This implies that the effects of each branch strongly depend on the task. Short-range attention is most useful when interactions remain within neighboring patches, whereas long-range attention is needed when information must travel farther. Fig. 8*(c)* supports this interpretation. After removing the short-range branch, long-range attention remains broad on other tasks but shifts toward nearby patches on SmokeGrid, suggesting that a dedicated short-range branch provides useful inductive bias for capturing local interactions.

Partition-of-unity blend. Fig. 9 shows the effect of the smooth partition-of-unity assembly in the decoder. In our Wendland variant, the weights $w_p(\boldsymbol{x})$ blend both the latent state before it enters the decoder MLP, $\bar{\boldsymbol{z}}(\boldsymbol{x}, t) = \sum_p w_p(\boldsymbol{x}) \boldsymbol{z}_p(t)$, and the generated patch-specific decoder matrices, $\boldsymbol{A}^{(\ell)}(\boldsymbol{x}, t) = \sum_p w_p(\boldsymbol{x}) \boldsymbol{A}_p^{(\ell)}(t)$. The hard-assignment variant instead sets these quantities to $\boldsymbol{z}_{p^\star}(t)$ and $\boldsymbol{A}_{p^\star}^{(\ell)}(t)$, where $p^\star$ is the patch nearest to $\boldsymbol{x}$. It therefore retains a distinct decoder for every patch but switches abruptly between them at patch boundaries. The two variants achieve similar rollout

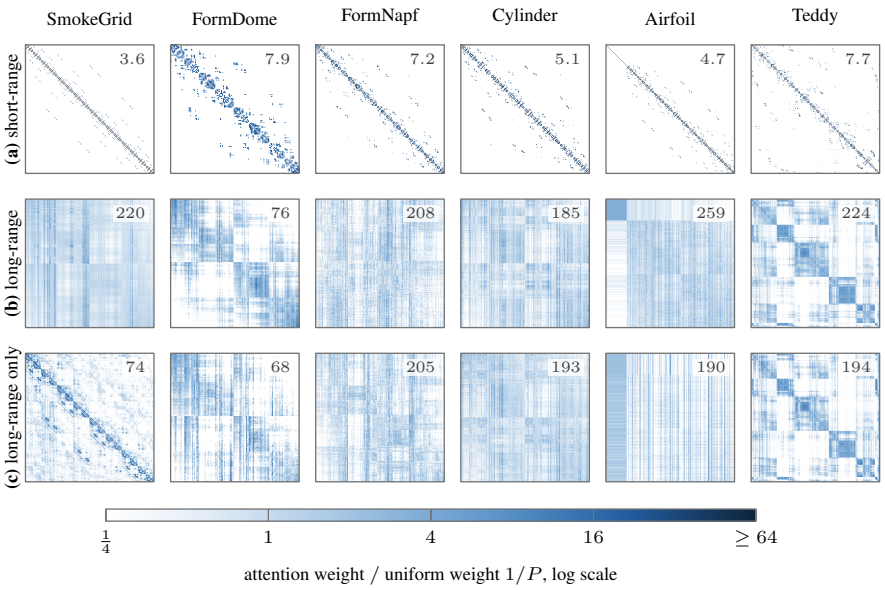


Figure 8: **Short-range and long-range attention.** Patch-to-patch attention averaged over heads for one trained run of each model. Rows show *(a)* short-range and *(b)* long-range attention in the full model, and *(c)* long-range attention after removing the short-range branch. Spatially neighboring patches have nearby indices, so their interactions appear near the diagonal. Colors show attention relative to uniform attention $1/P$ on a logarithmic scale. Each panel reports the effective number of attended patches, computed as the exponential of the mean row entropy.

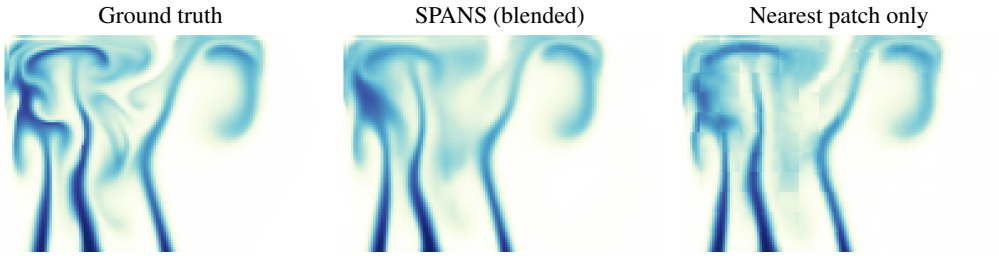


Figure 9: **Partition-of-unity blending.** SmokeGrid density at frame 20. Nearest-patch decoding produces visible seams along the 16×16 patch grid despite similar rollout MSE. All panels share one colour scale, and the source region at the bottom is cropped.

MSE (0.231×10^{-2} and 0.229×10^{-2}), yet hard assignment produces clearly visible seams. The smooth blend therefore provides continuity while retaining the patch-specific decoder.

Patch-conditioned decoder. Table 13a studies the complementary choice: the smooth partition-of-unity blend is retained, but the patch-conditioned decoder is replaced by a single decoder shared across all patches, i.e., $\boldsymbol{A}_p = \boldsymbol{A}$ for all patches p. The shared decoder remains continuous in space but applies the same transformation throughout the domain, reducing its expressive power. Sharing the decoder increases rollout error on Cylinder, Airfoil, and Teddy, while the errors on the remaining tasks are largely unchanged. The local-structure metrics in Table 13b further show that the shared decoder reconstructs local structures less accurately on the solid-mechanics tasks, where both the correlation and amplitude metrics decrease.

Together, these ablations show that partition-of-unity blending provides continuity across patch boundaries, whereas patch-dependent parameters enhance the local expressivity.

Table 13: **Patch-conditioned and shared decoders.** *(a)* Rollout MSE and its ratio to the patch-conditioned model. *(b)* Local-structure metrics (Appendix G). We report mean ± standard deviation over three seeds.

(a) Rollout MSE

Task	Patch-cond.	Shared	Ratio
SmokeGrid ($\times 10^{-2}$)	0.231 ± 0.011	0.232 ± 0.006	$1.00\times$
SmokeObs ($\times 10^{-2}$)	0.443 ± 0.012	0.456 ± 0.001	$1.03\times$
FormNapf	0.799 ± 0.016	0.834 ± 0.025	$1.04\times$
FormDome	0.922 ± 0.044	0.926 ± 0.052	$1.00\times$
Cylinder ($\times 10^{-1}$)	0.240 ± 0.026	0.352 ± 0.009	$1.47\times$
Airfoil ($\times 10^{-1}$)	0.159 ± 0.010	0.233 ± 0.043	$1.47\times$
Teddy ($\times 10^{-3}$)	0.633 ± 0.015	0.702 ± 0.015	$1.11\times$

(b) Local structure

Task	Metric	Patch-cond.	Shared
SmokeGrid	r_{HF}	0.952 ± 0.006	0.949 ± 0.022
SmokeObs	r_{HF}	0.772 ± 0.046	0.764 ± 0.017
FormNapf	ρ_z	0.448 ± 0.002	0.418 ± 0.001
	a_z	0.613 ± 0.005	0.584 ± 0.013
FormDome	ρ_z	0.403 ± 0.008	0.379 ± 0.004
	a_z	0.464 ± 0.018	0.474 ± 0.026
Teddy	ρ_{curv}	0.941 ± 0.007	0.913 ± 0.013
	a_{curv}	0.954 ± 0.008	0.920 ± 0.021

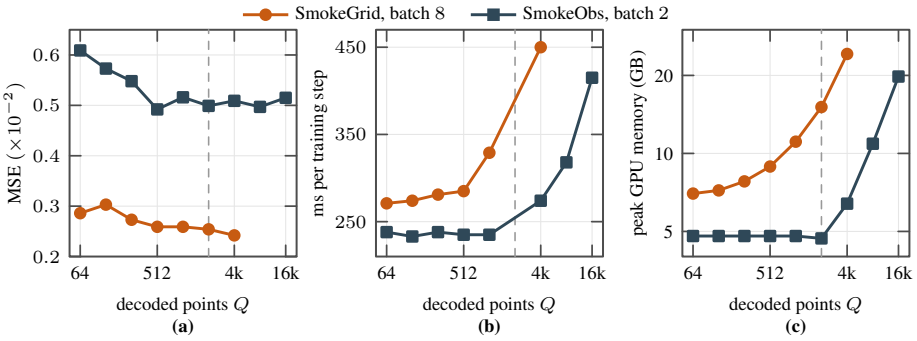


Figure 10: **Decode-subsampling ablation.** All settings are evaluated at the same training step. *(a)* Rollout MSE, *(b)* training time per step, and *(c)* peak GPU memory as the number of supervised query points Q is increased. The dashed line denotes the default $Q = 2048$. SmokeGrid and SmokeObs use batch sizes 8 and 2, respectively, so runtime and memory curves are compared only within each task. The two largest SmokeGrid settings exceed the available GPU memory.

Decode subsampling. To evaluate the sampled-query training strategy described in Appendix C.3, we vary the number of supervised query points Q on both smoke tasks and report the resulting accuracy–cost trade-off in Fig. 10. Across both tasks, increasing Q changes rollout error only moderately but substantially raises training time and peak memory. On SmokeGrid, the feasible settings increase training time per step by about $1.7\times$ and peak memory by $3.5\times$, while the two largest tested configurations exceed GPU memory. Because the tasks use different batch sizes, their absolute costs are not directly comparable; nevertheless, the same within-task trend motivates our choice of $Q = 2048$ as a conservative balance between supervision density and training cost.

H.5 Computational cost

In this section, we analyze the computational cost of SPANS, and compare it with other baselines. We measure three quantities: (1) Peak memory usage during training; (2) Rollout memory usage; (3) Rollout time per trajectory over the task's horizon T. We omit training time as a metric since each method has different training speed due to varying optimization strategies (e.g., one-step vs. multi-step objectives, different number of training steps, different optimizers). All measurements use one NVIDIA A100 GPU with 40 GB of memory at batch size 1.

Computational cost. First, we extend the reported rollout time measurement of SmokeGrid to include additional methods for completeness. Table 14 shows that overall, increasing the grid size from 128^2 to 512^2 the rollout memory usage less significantly compared to the closest latent baseline CALM-PDE (in terms of architectural design). In addition, the growth of rollout time of SPANS across different resolutions is relatively small, and comparable to that of other latent methods such as CALM-PDE, GAOT, and UPT. Grid-based methods (FNO, LOGLO-FNO, and CALM-CNN), on

Table 14: **Computational cost** on SmokeGrid with increasing resolution under one A100-40GB at batch size 1. Groups follow Table 1. OOM marks a configuration that exceeded the device. Training follows each method's own objective.

		training memory (GiB)				*rollout memory (GiB)*				*rollout time (s)*			
	Method	**128**	**256**	**384**	**512**	**128**	**256**	**384**	**512**	**128**	**256**	**384**	**512**
(I)	FNO	1.27	4.47	9.83	17.28	0.19	0.24	0.33	0.55	0.04	0.04	0.08	0.13
	LOGLO-FNO	2.02	7.46	16.54	29.23	0.19	0.24	0.38	0.64	0.05	0.06	0.16	0.29
(II)	GAOT	0.50	1.27	2.71	4.73	0.15	0.32	0.69	1.21	0.22	0.42	0.75	1.25
	Transolver	OOM	OOM	OOM	OOM	0.19	0.60	1.28	2.24	1.19	4.72	10.50	18.77
(III)	UPT	3.57	3.63	3.72	3.84	0.09	0.24	0.50	0.86	0.21	0.38	0.75	1.01
	CALM-PDE	14.33	OOM	OOM	OOM	0.71	2.78	6.22	11.05	0.24	0.61	1.24	2.00
	CALM-CNN	0.54	1.48	3.04	5.22	0.02	0.17	0.21	0.26	0.05	0.05	0.05	0.05
(IV)	IGNS	6.19	24.65	OOM	OOM	0.74	2.93	6.66	11.78	0.09	0.29	0.64	1.13
	MGN	8.94	35.78	OOM	OOM	1.18	4.60	10.40	18.32	1.12	4.26	9.54	16.96
	SPANS (ours)	1.61	2.75	2.98	3.02	0.15	0.57	1.31	2.44	0.10	0.36	0.85	1.71

the other hand, are the fastest, which might be attributed to the highly optimized CUDA kernels for convolutions and FFTs.

Fig. 11(a)–(g) shows the accuracy–cost trade-off separately across all seven tasks. On six tasks, no baseline lies in the shaded bottom-left region: the faster methods (FNO, LOGLO-FNO, and IGNS) are less accurate, whereas the methods closest in accuracy to SPANS (CALM-PDE and GAOT) are slower. The only exception is Cylinder in panel *(e)*, where the one-step Transolver variant is both faster and more accurate, consistent with Table 1.

Panel *(h)* reproduces the aggregate comparison in Fig. 3(a). For a method m, let $\mathcal{D}_m$ denote the tasks on which it was evaluated and let $v_{m,d}$ denote either its rollout time or rollout-averaged MSE on task d. Each axis reports the corresponding geometric-mean ratio to SPANS:

$$G_m(v) = \left(\prod_{d \in \mathcal{D}_m} \frac{v_{m,d}}{v_{\text{ours},d}} \right)^{1/|\mathcal{D}_m|} . \tag{76}$$

This normalization removes task-specific scales and places SPANS at $(1, 1)$. Every baseline lies above the horizontal line, while the three methods left of the vertical line trade accuracy for speed. Note that the set of tasks may differ across methods; e.g., FNO and LOGLO-FNO are applied only to grid-based tasks, whereas IGNS and MGN are evaluated only on the four fluid tasks.

Against the solver. We additionally compare the rollout time of SPANS with the reference ABAQUS solver that generated the data. For the FormNapf benchmark, SPANS runs in 26 ms per trajectory, compared to 70 minutes in ABAQUS. For the FormDome benchmark, SPANS runs in 11 ms per trajectory, compared to 14 minutes in ABAQUS. Note that the reference ABAQUS solver runs on a CPU, while SPANS runs on a GPU. These results highlight the efficiency of using a latent neural simulator for fast inference on complex physics problems where the reference solver is expensive.

Cost decomposition. Fig. 12 shows that the processor itself remains inexpensive across methods and resolutions, requiring only 1–2 ms and less than 0.14 GiB per step. Most resolution-dependent cost instead comes from mapping between the physical field and the latent representation. The encoder of SPANS takes only 3 ms, compared with 87 ms for CALM-PDE, while decoding the queried physical field is the largest stage of our model. This decomposition also motivates supervising only sampled query points during training, as studied in Appendix H.4.

I Qualitative comparisons

We provide qualitative comparisons on representative test trajectories to complement the rollout metrics and examine how well each method preserves spatial structure over time. The full set of rollout videos is available on the supplementary website. Across smoke transport, composite forming, vortex shedding, airfoil flow, and contact-induced deformation on Teddy, SPANS generally preserves the characteristic local structures of the dynamics, although some smoothing due to the MSE loss is

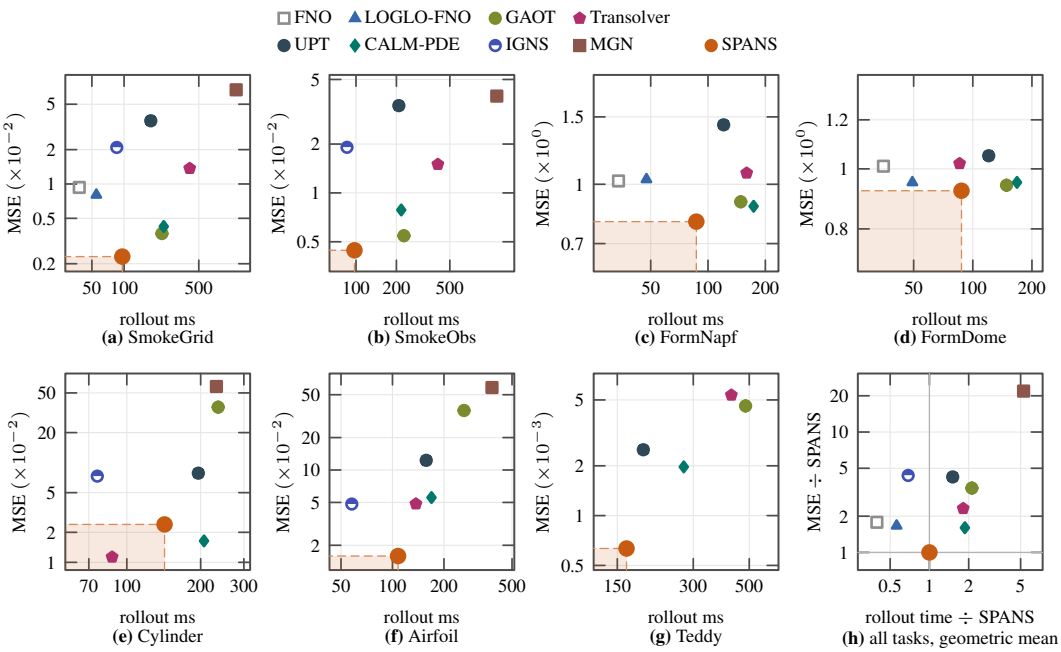


Figure 11: **Accuracy against rollout cost on every task.** *(a)–(g)* Rollout-averaged MSE versus the time to roll out one trajectory on a single A100 GPU at batch size one. Lower and farther left is better; the shaded bottom-left region is both faster and more accurate than SPANS. Methods are omitted where not applicable. *(h)* Aggregate comparison from Fig. 3(a), with each axis showing the geometric-mean ratio to SPANS over the evaluated tasks (Eq. (76)). SPANS lies at $(1, 1)$, and the gray lines mark equal cost and error.

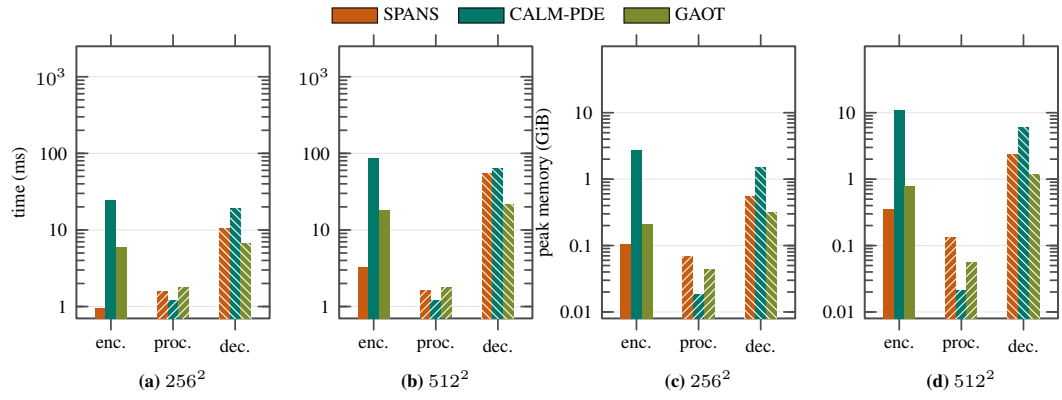


Figure 12: **Per-stage computational cost.** Wall-clock time and peak memory for the encoder, one processor step, and the decoder at 256^2 and 512^2. The encoder runs once per trajectory, the processor once per rollout step, and the decoder once per output frame. *(a,b)* Wall-clock time. *(c,d)* Peak memory.

visible in the predictions. These visualizations support the quantitative results and suggest that the structured patch representation retains much of the local detail under compression.

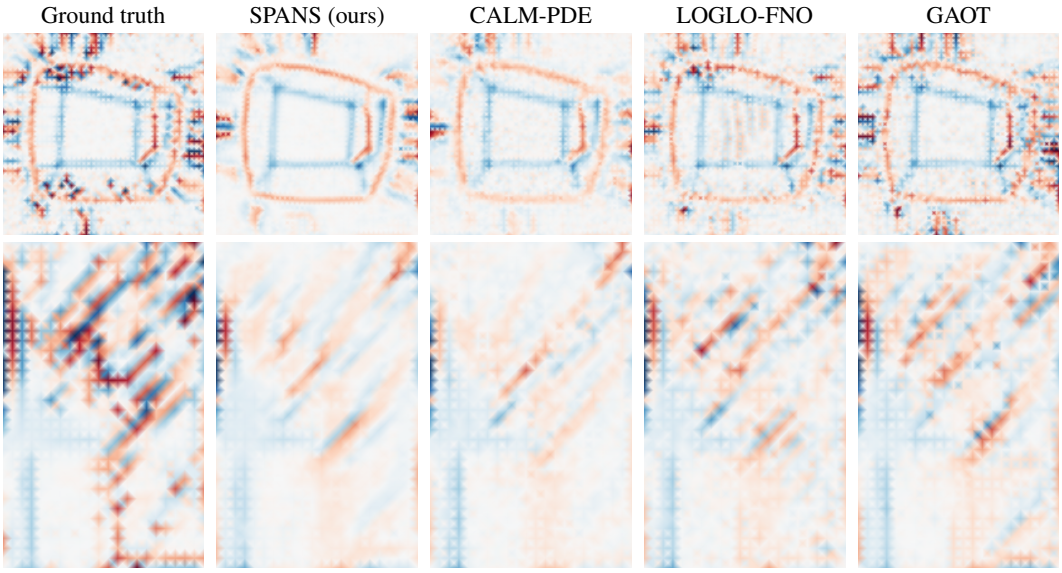


Figure 13: **Forming qualitative comparison.** Final-frame out-of-plane high-band displacement of the first ply for the median-error test trajectory of each task, ranked by SPANS's rollout error. Rows show FormNapf and FormDome, each with a shared symmetric colour scale.

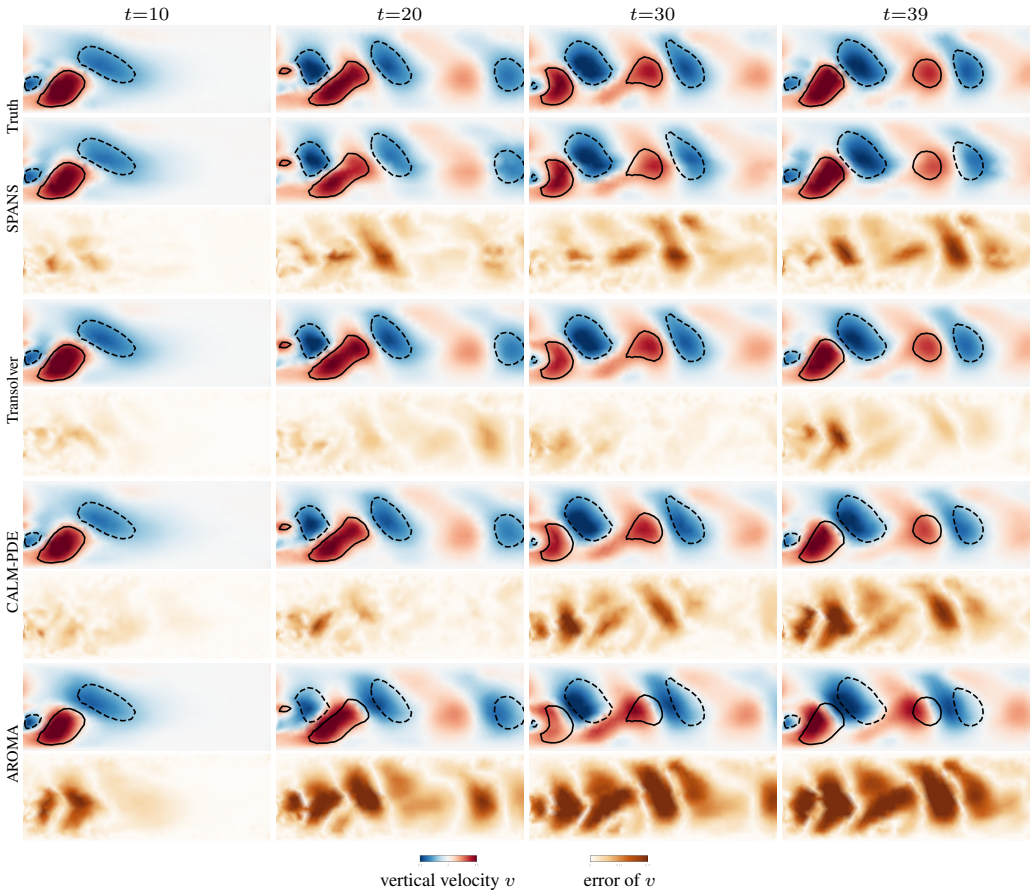


Figure 14: **Cylinder qualitative comparison.** Strong-wake test trajectory, cropped to two diameters upstream of the cylinder. For each model, the upper row shows the predicted vertical velocity v, while the lower row shows the corresponding absolute error relative to the reference solution. Dashed contours indicate the reference ground-truth vortex locations. Transolver closely follows the reference wake evolution, while SPANS and CALM-PDE preserves the main vortex structure, but with slightly larger phase errors. AROMA drifts significantly from the reference wake evolution.

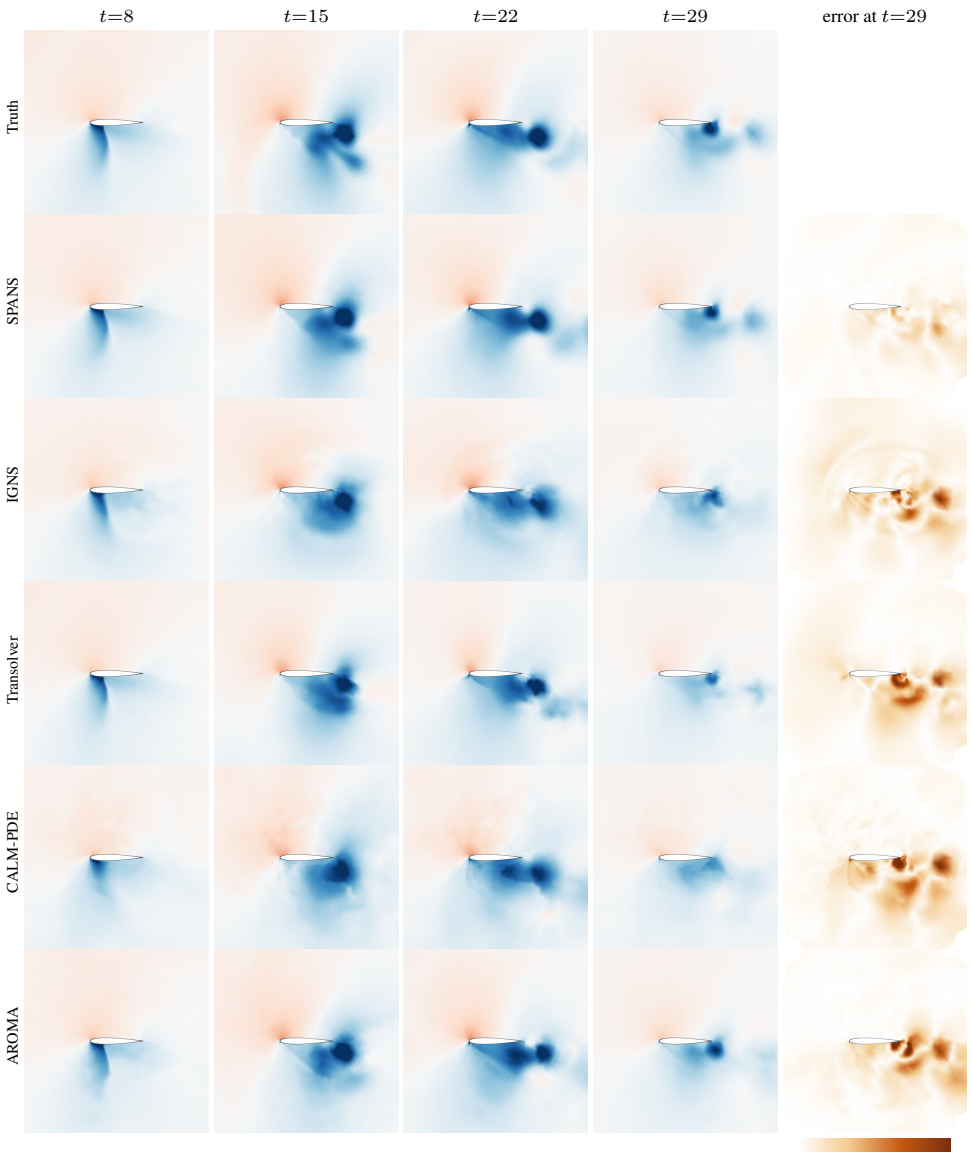


Figure 15: **Airfoil qualitative comparison.** Test trajectory with the largest frozen-frame error, shown in a local region around the wing. The first four columns show the density deviation from the free stream on a shared symmetric scale. The final column shows the absolute density error at $t = 29$ on a common scale across all methods, where darker values indicate larger errors.

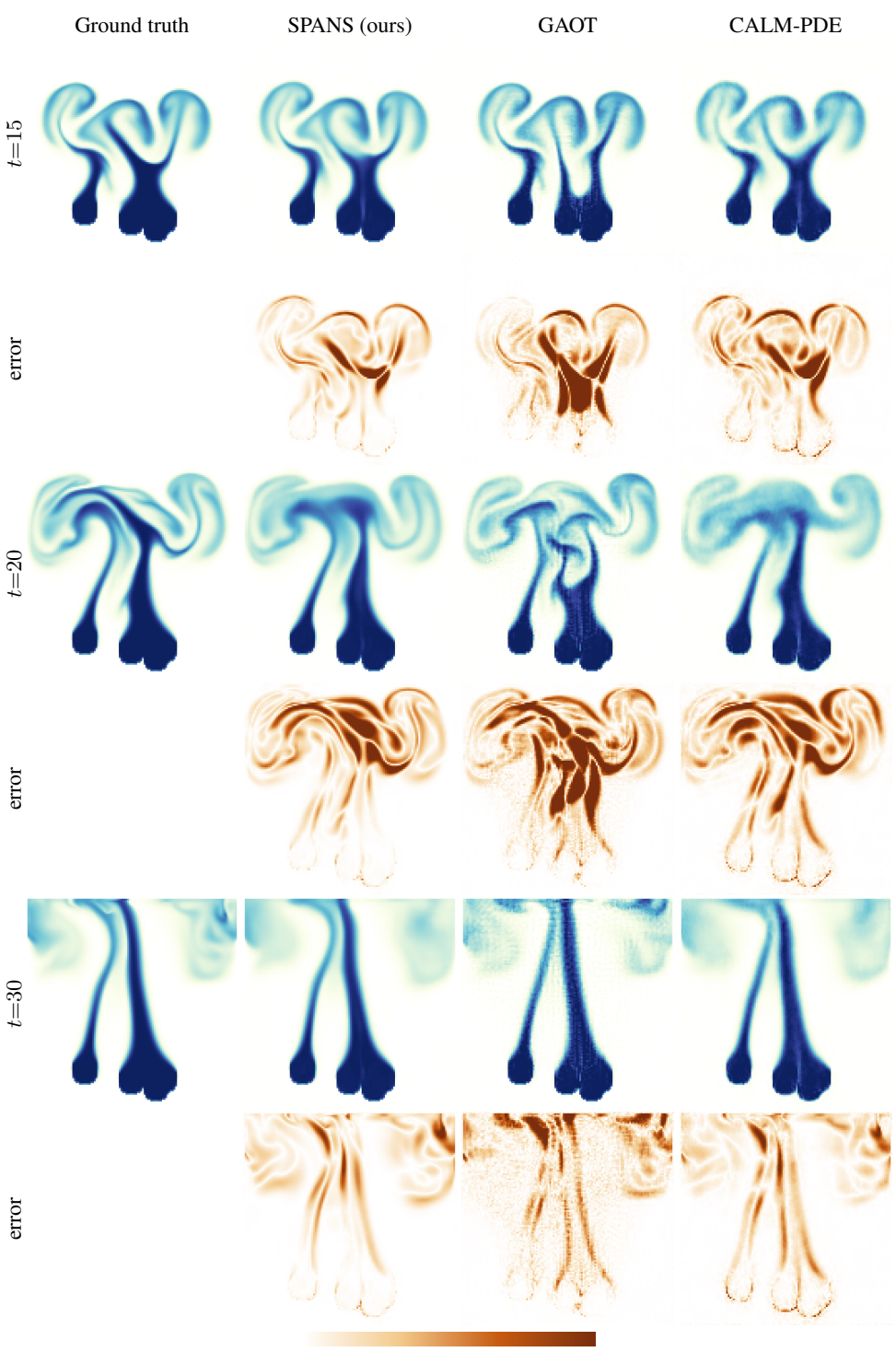


Figure 16: **SmokeGrid qualitative comparison.** Test trajectory with the largest frozen-frame error, shown at t=15, 20, and 30. Each prediction is followed by absolute error on one shared scale; darker values are worse.

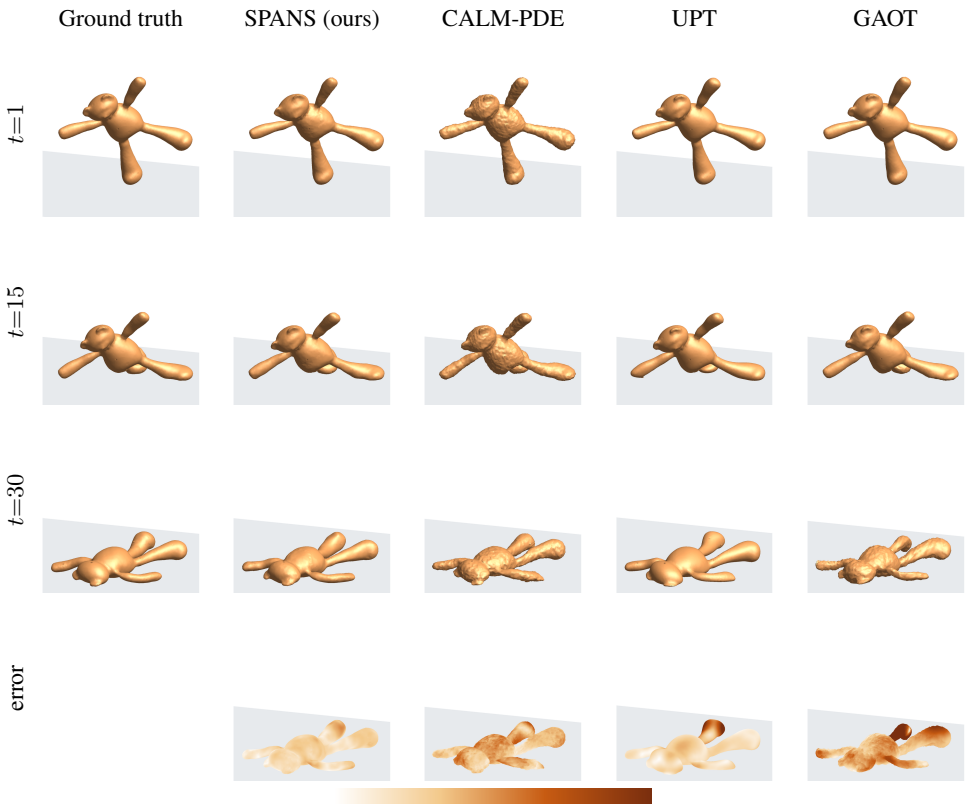


Figure 17: **Teddy qualitative comparison.** Test trajectory with the largest frozen-frame error, shown at t=1, 15, and 30. The bottom row shows per-point displacement error at t=30 on one shared scale; darker values are worse. Columns are ordered by rollout error; UPT uses our second-order latent update (Appendix E).