Learning to Accelerate Partial Differential Equations via Latent Global Evolution

Tailin Wu
Department of Computer Science
Stanford University
tailin@cs.stanford.edu

Takashi Maruyama
NEC Corp. & Stanford University
49takashi@nec.com &
takashi279@cs.stanford.edu

Jure Leskovec
Department of Computer Science
Stanford University
jure@cs.stanford.edu

Abstract

Simulating the time evolution of Partial Differential Equations (PDEs) of large-scale systems is crucial in many scientific and engineering domains such as fluid dynamics, weather forecasting and their inverse optimization problems. However, both classical solvers and recent deep learning-based surrogate models are typically extremely computationally intensive, because of their local evolution: they need to update the state of each discretized cell at each time step during inference. Here we develop Latent Evolution of PDEs (LE-PDE), a simple, fast and scalable method to accelerate the simulation and inverse optimization of PDEs. LE-PDE learns a compact, global representation of the system and efficiently evolves it fully in the latent space with learned evolution models. LE-PDE achieves speed-up by having a much smaller latent dimension to update during long rollout as compared to updating in the input space. We introduce new learning objectives to effectively learn such latent dynamics to ensure long-term stability. We further introduce techniques for speeding up inverse optimization of boundary conditions for PDEs via backpropagation through time in latent space, and an annealing technique to address the non-differentiability and sparse interaction of boundary conditions. We test our method in a 1D benchmark of nonlinear PDEs, 2D Navier-Stokes flows into turbulent phase and an inverse optimization of boundary conditions in 2D Navier-Stokes flow. Compared to other strong baselines, we demonstrate up to $128\times$ reduction in the dimensions to update, and up to $15\times$ improvement in speed, while achieving competitive accuracy. [1].

1 Introduction

Many problems across science and engineering are described by Partial Differential Equations (PDEs). Among them, temporal PDEs are of huge importance. They describe how the state of a (complex) system evolves with time, and numerically evolving such equations are used for forward prediction and inverse optimization across many disciplines. Example application includes weather forecasting [1], jet engine design [2], nuclear fusion [3], laser-plasma interaction [4], astronomical simulation [5], and molecular modeling [6].

To numerically evolve such PDEs, decades of works have yielded (classical) PDE solvers that are tailored to each specific problem domain [7]. Albeit principled and accurate, classical PDE solvers

[1] Project website and code can be found at http://snap.stanford.edu/le_pde/.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

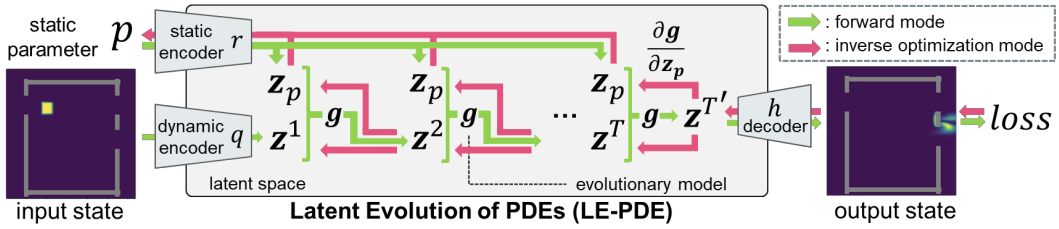


Figure 1: LE-PDE schematic. In forward mode (green), LE-PDE evolves the dynamics in a global latent space. In inverse optimization mode (red), it optimizes parameter p (*e.g.* boundary) through latent unrolling. The compressed latent vector and dynamics can significantly speed up both modes.

are typically slow due to the small time steps or implicit method required for numerical stability, and their time complexity typically scales linearly or super-linearly with the number of cells the domain is discretized into [8]. For practical problems in science and engineering, the number of cells at each time step can easily go up to millions or billions and may even require massively parallel supercomputing resources [9, 10]. Besides forward modeling, inverse problems, such as inverse optimization of system parameters and inverse parameter inference, also share similar scaling challenge [11]. How to effectively speed up the simulation while maintaining reasonable accuracy remains an important open problem.

Recently, deep learning-based surrogate models have emerged as attractive alternative to complement [12] or replace classical solvers [13, 14]. They directly learn the dynamics from data and alleviate much engineering effort. They typically offer speed-up due to explicit forward mapping [15, 16], larger time intervals [14], or modeling on a coarser grid [12, 17]. However, their evolution scales with the discretization, since they typically need to update the state of each discretized cell at each time step, due to the local nature of PDEs [18]. For example, if a problem is discretized into 1 million cells, deep learning-based surrogate models (*e.g.*, CNNs, Graph Networks, Neural Operators) will need to evolve these 1 million cells per time step. How to go beyond updating each individual cells and further speed up such models remains a challenge.

Here we present Latent Evolution of PDEs (LE-PDE) (Fig. 1), a simple, fast and scalable method to accelerate the simulation and inverse optimization of PDEs. Our key insight is that a common feature of the dynamics of many systems of interest is the presence of dominant, low-dimensional coherent structures, suggesting the possibility of efficiently evolving the system in a low-dimensional global latent space. Based on this observation, we develop LE-PDE, which learns the evolution of dynamics in a global latent space. Here by "global" we mean that the dimension of the latent state is fixed, instead of scaling linearly with the number of cells as in local models. LE-PDE consists of a dynamic encoder that compresses the input state into a dynamic latent vector, a static encoder that encodes boundary conditions and equation parameters into a static latent vector, and a latent evolution model that evolves the dynamic latent vector fully in the latent space, and decode via a decoder only as needed. Although the idea of latent evolution has appeared in other domains, such as in computer vision [19, 20, 21] and robotics [22, 23, 24, 25], these domains typically have clear object structure in visual inputs allowing compact representation. PDEs, on the other hand, model dynamics of continuum (*e.g.*, fluids, materials) with infinite dimensions, without a clear object structure, and sometimes with chaotic turbulent dynamics, and it is pivotal to model their long-term evolution accurately. Thus, learning the latent dynamics of PDEs presents unique challenges.

We introduce a multi-step latent consistency objective, to encourage learning more stable long-term evolution in latent space. Together with the multi-step loss in the input space, they encourage more accurate long-term prediction. To accelerate inverse optimization of PDEs which is pivotal in engineering (*e.g.* optimize the boundary condition so that the evolution of the dynamics optimizes certain predefined objective), we show that LE-PDE can allow faster optimization, via backpropagation through time in latent space instead of in input space. To address the challenge that the boundary condition may be non-differentiable or too sparse to receive any gradient, we design an annealing technique for the boundary mask during inverse optimization.

We demonstrate our LE-PDE in standard PDE-learning benchmarks of a 1D family of nonlinear PDEs and a 2D Navier-Stokes flow into turbulent phase, and design an inverse optimization problem in 2D Navier-Stokes flow to probe its capability. Compared with state-of-the-art deep learning-based

surrogate models and other strong baselines, we show up to 128× reduction in the dimensions to update and up to 15× speed-up compared to modeling in input space, and competitive accuracy.

2 Problem Setting and Related Work

We consider temporal Partial Differential Equations (PDEs) w.r.t. time $t \in [0, T]$ and multiple spatial dimensions $\mathbf{x} = [x_1, x_2, ...x_D] \in \mathbb{X} \subseteq \mathbb{R}^D$. We follow similar notation as in [7]. Specifically,

$$\partial_t \mathbf{u} = F(\mathbf{x}, \mathbf{a}, \mathbf{u}, \partial_{\mathbf{x}}\mathbf{u}, \partial_{\mathbf{xx}}\mathbf{u}, ...), \qquad (t, \mathbf{x}) \in [0, T] \times \mathbb{X}, \tag{1}$$

$$\mathbf{u}(0, x) = \mathbf{u}^0(\mathbf{x}),\ B[\mathbf{u}](t, \mathbf{x}) = 0, \qquad \mathbf{x} \in \mathbb{X}, (t, \mathbf{x}) \in [0, T] \times \partial\mathbb{X}. \tag{2}$$

Here $\mathbf{u} : [0, T] \times \mathbb{X} \to \mathbb{R}^n$ is the solution, which is an infinite-dimensional function. $\mathbf{a}$ is time-independent static parameters of the system, which can be defined on each location $\mathbf{x}$, *e.g.* diffusion coefficient that varies in space but static in time, or a global parameter. F is a linear or nonlinear function on the arguments of $(\mathbf{x}, \mathbf{a}, \mathbf{u}, \partial_{\mathbf{x}}\mathbf{u}, \partial_{\mathbf{xx}}\mathbf{u}, ...)$. Note that in this work we consider time-independent PDEs where F does not explicitly depend on t. $\mathbf{u}^0(\mathbf{x})$ is the initial condition, and $B[\mathbf{u}](t, \mathbf{x}) = 0$ is the boundary condition when $\mathbf{x}$ is on the boundary of the domain $\partial\mathbb{X}$ across all time $t \in [0, T]$. Here $\partial_{\mathbf{x}}\mathbf{u} = \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$, $\partial_{\mathbf{xx}}\mathbf{u} = \frac{\partial^2 \mathbf{u}}{\partial \mathbf{x}^2}$ are first- and second-order partial derivatives, which are a matrix and a 3-order tensor, respectively (since $\mathbf{x}$ is a vector). *Solving* such temporal PDEs means computing the state $\mathbf{u}(t, \mathbf{x})$ for any time $t \in [0, T]$ and location $\mathbf{x} \in \mathbb{X}$ given the above initial and boundary conditions.

Classical solvers for solving PDEs. To numerically solve the above PDEs, classical numerical solvers typically discretize the domain $\mathbb{X}$ into a finite grid or mesh $X = \{c_i\}_{i=1}^N$ with N non-overlapping cells. Then the infinite-dimensional solution function of $\mathbf{u}(t, \mathbf{x})$ is discretized into $U^k = \{\mathbf{u}_i^k\}_{i=1}^N \in \mathbb{U}$ for each cell i and time $t = t_k, k = 1, 2, ...K$. $\mathbf{a}$ is similarly discretized into $\{a_i\}_{i=1}^N$ with values in each cell. Mainstream numerical methods, including Finite Difference Method (FDM) and Finite Volume Method (FVM), proceed to evolve such temporal PDEs by solving the equation at state $\{\mathbf{u}_i^{k+1}\}$ at time $t = t_{k+1}$ from state $\{\mathbf{u}_i^k\}$ at time t_k. These solvers are typically slow due to small time/space intervals required for numerical stability, and needing to update each cell at each time steps. For more detailed information on classical solvers, see Appendix A

Deep learning-based surrogate modeling. There are two main approaches in deep learning-based surrogate modeling. The first class of method is autoregressive methods, which learns the mapping f_θ with parameter θ of the discretized states U^k between consecutive time steps t_k and t_{k+1}: $\hat{U}^{k+1} = f_\theta(\hat{U}^k, p), k = 0, 1, 2, ...$ Here $\hat{U^k} = \{\hat{\mathbf{u}}_i^k\}_{i=1}^N$ is the model f_θ's predicted state for $U^k = \{\mathbf{u}_i^k\}_{i=1}^N$ at time t_k, with $\hat{U^0} := U^0$. $p = (\partial\mathbb{X}, \{a_i\}_{i=1}^N)$ is the system parameter which includes the boundary domain $\partial\mathbb{X}$ and discretized static parameters $\{a_i\}_{i=1}^N$. Repetitively apply f_θ at inference time results in autoregressive rollout

$$\hat{U}^{k+m} = (f_\theta(\cdot, p))^{(m)}(\hat{U}^k), m \geq 1. \tag{3}$$

Here $f_\theta(\cdot, p) : \mathbb{U} \to \mathbb{U}$ is a partial function whose second argument is fulfilled by the static system parameter p. Typically f_θ is modeled using CNNs (if the domain $\mathbb{X}$ is discretized into a grid), Graph Neural Networks (GNNs, if the domain $\mathbb{X}$ is discretized into a mesh). These methods all involve local computation, where the value u_i^{k+1} at cell i at time t_{k+1} depend on its neighbors $\{u_j^k\}_{j \in \mathcal{N}(i)}$ at time t_k, where $\mathcal{N}(i)$ is the set of neighbors up to certain hops. Such formulation includes CNN-based models [26], GNN-based models [7, 27, 28] and their hierarchical counterparts [18, 29]. The surrogate modeling with local dynamics makes sense, since the underlying PDE is essentially a local equation that stipulates how the solution function $\mathbf{u}$'s value at location $\mathbf{x}$ depends on the values at its infinitesimal neighborhood. The second class of method is Neural Operators [14, 30, 31, 32, 33, 34, 35, 36], which learns a neural network (NN) that approximates a mapping between infinite-dimensional functions. Although having the advantage that the learned mapping is discretization invariant, given a specific discretization, Neural Operators still needs to update the state at each cell based on neighboring cells (and potentially cells far away), which is still inefficient at inference time, especially dealing with larger-scale problems. In contrast to the above classes of deep learning-based approaches that both requires local evolution at inference time, our LE-PDE method focuses on improving efficiency. Using a learned global latent space, LE-PDE removes the need for local evolution and can directly evolve the system dynamics via a global latent vectors $\mathbf{z}^k \in \mathbb{R}^{d_z}$ for time t_k. This offers great potential for speed-up due to the significant reduction in representation.

Inverse optimization. Inverse optimization is the problem of optimizing the parameters p of the PDE, including boundary $\partial\mathbb{X}$ or static parameter $\mathbf{a}$ of the equation, so that a predefined objective $L_d[\mathbf{a}, \partial\mathbb{X}] = \int_{t=t_s}^{t_e} \ell_d[\mathbf{u}(t, \mathbf{x})]dt$ is minimized. Here the state $\mathbf{u}(t, \mathbf{x})$ implicitly depends on $\mathbf{a}, \partial\mathbb{X}$ through the PDE (Eq. 1) and the boundary condition (Eq. 2). Such problems have huge importance in engineering, *e.g.* in designing jet engines [2] and materials [37] where the objective can be minimizing drag or maximizing durability, and inverse parameter inference (*i.e.* history matching) [38, 39, 40] where the objective can be maximum a posteriori estimation. To solve such problem, classical methods include adjoint method [41, 42], shooting method [43], collocation method [44], etc. One recent work [45] explores optimization via backpropagation through differential physics in the input space, demonstrating speed-up and improved accuracy compared to classical CEM method [46]. However, for long rollout and large input size, the computation becomes intensive to the point of needing to save gradients in files. In comparison, LE-PDE allows backpropagation in latent space, and due to the much smaller latent dimension and evolution model, it can significantly reduce the time complexity in inverse optimization.

Reduced-order modeling. A related class of work is reduced-order modeling. Past efforts typically use linear projection into certain basis functions [47, 48, 49, 50, 51, 52, 53, 54] which may not have enough representation power. A few recent works explore NN-based encoding [55, 56, 57, 58, 59, 60] for fluid modeling. Compared to the above works, we focus on speeding up simulation and inverse optimization of more general PDEs using expressive NNs, with novel objectives, and demonstrate competitive performance compared to state-of-the-art deep learning-based models for PDEs.

3 Our approach LE-PDE

In this section, we detail our Latent Evolution of Partial Differential Equations (LE-PDE) method. We first introduce the model architecture (Sec. 3.1, and then we introduce learning objective to effectively learn faithfully long-term evolution (Sec. 3.2). In Sec. 3.3, we introduce efficient inverse optimization in latent space endowed by our method.

3.1 Model architecture

The model architecture of LE-PDE consists of four components: (1) a dynamic encoder $q : \mathbb{U} \to \mathbb{R}^{d_z}$ that maps the input state $U^k = \{\mathbf{u}_i^k\}_{i=1}^N \in \mathbb{U}$ to a latent vector $\mathbf{z}^k \in \mathbb{R}^{d_z}$; (2) an (optional) static encoder $r : \mathbb{P} \to \mathbb{R}^{d_{zp}}$ that maps the (optional) system parameter $p \in \mathbb{P}$ to a static latent embedding $\mathbf{z}_p$; (3) a decoder $h : \mathbb{R}^{d_z} \to \mathbb{U}$ that maps the latent vector $\mathbf{z}^k \in \mathbb{R}^{d_z}$ back to the input state U^k; (4) a latent evolution model $g : \mathbb{R}^{d_z} \times \mathbb{R}^{d_{zp}} \to \mathbb{R}^{d_z}$ that maps $\mathbf{z}^k \in \mathbb{R}^{d_z}$ at time t_k and static latent embedding $\mathbf{z}_p \in \mathbb{R}^{d_{zp}}$ to $\mathbf{z}^{k+1} \in \mathbb{R}^{d_z}$ at time t_{k+1}. We employ the temporal bundling trick [7] where each input state U^k can include states over a fixed length S of consecutive time steps, in which case each latent vector $\mathbf{z}_k$ will encode such bundle of states, and each latent evolution will predict the latent vector for the next bundle of S steps. S is a hyperparameter and may be chosen depending on the problem, and $S = 1$ reduces to no bundling. A schematic of the model architecture and its inference is illustrated in Fig. 1. Importantly, we require that for the dynamic encoder q, it needs to have a flatten operation and MultiLayer Perception (MLP) head that maps the feature map into a single fixed-length vector $\mathbf{z} \in R^{d_z}$. In this way, the dimension of the latent space does not scale linearly with the dimension of the input, which has the potential to significantly compress the input, and can make the long-term prediction much more efficient. At inference time, LE-PDE performs autoregressive rollout in latent space $\mathbb{R}^{d_z}$:

$$\hat{U}^{k+m} = h \circ g\left(\cdot, r(p)\right)^{(m)} \circ q(\hat{U}^k) \equiv h\Bigg(\underbrace{g(\cdot, r(p)) \circ ... \circ g(\cdot, r(p))}_{\text{composing } m \text{ times}}\Big(q(\hat{U}^k)\Big)\Bigg). \tag{4}$$

Compared to autoregressive rollout in input space (Eq. 3), LE-PDE can significantly improve efficiency with a much smaller dimension of $\mathbf{z}^k \in \mathbb{R}^{d_z}$ compared to $U^k \in \mathbb{U}$. Here we do not limit the architecture for encoder, decoder and latent evolution models. Depending on the input U^k, the encoder q and decoder h can be a CNN or GNN with a (required) MLP head. In this work, we focus on input that is discretized as grid, so the encoder and decoder are both CNN+MLP, and leave other architecture (*e.g.* GNN+MLP) for future work. For static encoder r, it can be a simple MLP if the system parameter p is a vector (*e.g.* equation parameters) or CNN+MLP if p is a 2D or 3D tensor (*e.g.* boundary mask, spatially varying diffusion coefficient). We model the latent evolution model g

as an MLP with residual connection from input to output. The architectures used in our experiments, are detailed in Appendix C, together with discussion of its current limitations.

3.2 Learning objective

Learning surrogate models that can faithfully roll out long-term is an important challenge. Given discretized inputs $\{U^k\}, k = 1, ...K + M$, we introduce the following objective to address it:

$$L = \frac{1}{K}\sum_{k=1}^{K}(L^k_{\text{multi-step}} + L^k_{\text{recons}} + L^k_{\text{consistency}}). \tag{5}$$

$$\text{where} \begin{cases} L^k_{\text{multi-step}} = \sum_{m=1}^{M} \alpha_m \ell(\hat{U}^{k+m}, U^{k+m}), \\ L^k_{\text{recons}} = \ell(h(q(U^k)), U^k) \\ L^k_{\text{consistency}} = \sum_{m=1}^{M} \frac{||g(\cdot, r(p))^{(m)} \circ q(U^k) - q(U^{k+m})||_2^2}{||q(U^{k+m})||_2^2} \end{cases} \tag{6}$$

Here ℓ is the loss function for individual predictions, which can typically be MSE or L2 loss. $\hat{U}^{k+m}$ is given in Eq. (4). L^k_{recons} aims to reduce reconstruction loss. $L^k_{\text{multi-step}}$ performs latent multi-step evolution given in Eq. (4) and compare with the target U^{k+m} in *input* space, up to time horizon M. α_m are weights for each time step, which we find that $(\alpha_1, \alpha_2, ...\alpha_M) = (1, 0.1, 0.1, ...0.1)$ works well. Besides encouraging better prediction in input space via $L^k_{\text{multi-step}}$, we also want a stable long-term rollout in latent space. This is because in inference time, we want to mainly perform autoregressive rollout in latent space, and decode to input space only when needed. Thus, we introduce a novel latent consistency loss $L^k_{\text{consistency}}$, which compares the m-step latent rollout $g(\cdot, r(p))^{(m)} \circ q(U^k)$ with the latent target $q(U^{k+m})$ in *latent* space. The denominator $||q(U^{k+m})||_2^2$ serves as normalization to prevent the trivial solution that the latent space collapses to a single point. Taken together, the three terms encourage a more accurate and consistent long-term evolution both in latent and input space. In Sec. 4.4 we will investigate the influence of $L^k_{\text{consistency}}$ and $L^k_{\text{multi-step}}$.

3.3 Accelerating inverse optimization

In addition to improved efficiency for forward simulation, LE-PDE also allows more efficient inverse optimization, via backpropagation through time (BPTT) in latent space. Given a specified objective $L_d[p] = \sum_{k=k_s}^{k_e} \ell(U^k)$ which is a discretized version of $L_d[\mathbf{a}, \partial\mathbb{X}]$ in Sec. 2, we define the objective:

$$L_d[p] = \sum_{m=k_s}^{k_e} \ell_d(\hat{U}^m(p)) \tag{7}$$

where $\hat{U}^m = \hat{U}^m(p)$ is given by Eq. (4) setting $k = 0$ using our learned LE-PDE, which starts at initial state of U^0, encode it and p into latent space, evolves the dynamics in latent space and decode to $\hat{U}^m$ as needed. The static latent embedding $\mathbf{z}_p = r(p)$ influences the latent evolution at each time step via $g(\cdot, r(p))$. An approximately optimal parameter p can then be found by computing gradients $\frac{\partial L_d[p]}{\partial p}$, using optimizers such as Adam [61] (The gradient flow is visualized as the red arrows in Fig. 1). When p is a boundary parameter, *e.g.* location of the boundary segments or obstacles, there is a challenge. Specifically, for CNN encoder q, the boundary information is typically provided as a binary mask indicating which cells are outside the simulation domain Ω. The discreteness of the mask prevents the backpropagation of the model. Moreover, the boundary cells may interact sparsely with the bulk, which can lead to vanishing gradient during inverse optimization. To address this, we introduce a function that maps p to a soft boundary mask with temperature, and during inverse optimization, anneal the temperature from high to low. This allows the gradient to pass through mask to p, and stronger gradient signal. For more information, see Appendix B.

4 Experiments

In the experiments, we aim to answer the following questions: (1) Does LE-PDE able to learn accurately the long-term evolution of challenging systems, and compare competitively with state-of-the-art methods? (2) How much can LE-PDE reduce representation dimension and improving speed, especially with larger systems? (3) Can LE-PDE improve and speed up inverse optimization? For the first and second question, since in general there is a fundamental tradeoff between compression

Table 1: Performance of models in 1D for scenarios **E1**,**E2**, **E3**. Accumulated error $= \frac{1}{n_x}\sum_{t,x}$ MSE. Representation dimension ($= S \times n_x$ here) is the number of dimensions to update at each time step. The bold values represent the best performance for experiments and underline shows second best.

		Accumulated Error ↓					**Runtime [ms] ↓**				**Representation dim ↓**	
	(n_t, n_x)	WENO5	FNO-RNN	FNO-PF	MP-PDE	**LE-PDE (ours)**	WENO5	MP-PDE	**LE-PDE full (ours)**	**LE-PDE evo (ours)**	MP-PDE	**LE-PDE (ours)**
E1	$(250, 100)$	2.02	11.93	**0.54**	1.55	<u>1.13</u>	1.9×10^3	90	20	**8**	2500	**128**
E1	$(250, 50)$	6.23	29.98	**0.51**	1.67	<u>1.20</u>	1.8×10^3	80	20	**8**	1250	**128**
E1	$(250, 40)$	9.63	10.44	**0.57**	1.47	<u>1.17</u>	1.7×10^3	80	20	**8**	1000	**128**
E2	$(250, 100)$	<u>1.19</u>	17.09	2.53	1.58	**0.77**	1.9×10^3	90	20	**8**	2500	**128**
E2	$(250, 50)$	5.35	3.57	2.27	<u>1.63</u>	**1.13**	1.8×10^3	90	20	**8**	1250	**128**
E2	$(250, 40)$	8.05	3.26	2.38	<u>1.45</u>	**1.03**	1.7×10^3	80	20	**8**	1000	**128**
E3	$(250, 100)$	4.71	10.16	5.69	<u>4.26</u>	**3.39**	4.8×10^3	90	19	**6**	2500	**64**
E3	$(250, 50)$	11.71	14.49	5.39	**3.74**	<u>3.82</u>	4.5×10^3	90	19	**6**	1250	**64**
E3	$(250, 40)$	15.94	20.90	5.98	**3.70**	<u>3.78</u>	4.4×10^3	90	20	**8**	1000	**128**

(reduction of dimensions to represent a state) and accuracy [62, 63], *i.e.* the larger the compression to improve speed, the more lossy the representation is, we will need to sacrifice certain amount of accuracy. Therefore, the goal of LE-PDE is to maintain a reasonable or competitive accuracy (maybe slightly underperform state-of-the-art), while achieving significant compression and speed up. Thus, to answer these two questions, we test LE-PDE in standard benchmarks of a 1D family of nonlinear PDEs to test its generalization to new system parameters (Sec. 4.1), and a 2D Navier-Stokes flow up to turbulent phase (Sec. 4.2). The PDEs in the above scenarios have wide and important application in science and engineering. In each domain, we compare LE-PDE's long-term evolution performance, speed and representation dimension with state-of-the-art deep learning-based surrogate models in the domain. Then we answer question (3) in Section 4.3. Finally, in Section 4.4, we investigate the impact of different components of LE-PDE and important hyperparameters.

4.1 1D family of nonlinear PDEs

Data and Experiments. In this section, we test LE-PDE's ability to generalize to unseen equations with different parameters in a given family. We use the 1D benchmark in [7], whose PDEs are

$$\left[\partial_t u + \partial_x(\alpha u^2 - \beta \partial_x u + \gamma \partial_{xx} u)\right](t, x) = \delta(t, x) \tag{8}$$

$$u(0, x) = \delta(0, x), \quad \delta(t, x) = \sum_{j=1}^{J} A_j \sin(\omega_j t + 2\pi \ell_j x/L + \phi_j) \tag{9}$$

Here the parameter $p = (\alpha, \beta, \gamma)$. The term δ is a forcing term [64] with $J = 5, L = 16$ and coefficients A_j and ω_j sampled uniformly from $A_j \sim U[-0.5, 0.5]$, $\omega_j \sim U[-0.4, 0.4]$, $\ell_j \in \{1, 2, 3\}$, $\phi_j \sim U[0, 2\pi)$. Space is uniformly discretized to $n_x = 200$ in $[0, 16)$ and time is uniformly discretized to $n_t = 250$ points in $[0, 4]$. Space and time are further downsampled to resolutions of $(n_t, n_x) \in \{(250, 100), (250, 50), (250, 40)\}$. The $\partial_x(\alpha u^2)$ advection term makes the PDE nonlinear. There are 3 scenarios with increasing difficulty: **E1:** Burgers' equation without diffusion $p = (1, 0, 0)$; **E2**: Burgers' equation with variable diffusion $p = (1, \eta, 0)$ where $\eta \in [0, 0.2]$; **E3**: mixed scenario with $p = (\alpha, \beta, \gamma)$ where $\alpha \in [0, 3], \beta \in [0, 0.4]$ and $\gamma \in [0, 1]$. **E1** tests the model's ability to generalize to new conditions with same equation. **E2** and **E3** test the model's ability to generalize to novel parameters of PDE with the same family. We compare LE-PDE with state-of-the-art deep learning-based surrogate models for this dataset, specifically MP-PDE [7] (a GNN-based model) and Fourier Neural Operators (FNO) [14]. For FNO, we compare with two versions: FNO-RNN is the autoregressive version in Section 5.3 of their paper, trained with autoregressive rollout; FNO-PF is FNO improved with the temporal bundling and push-forward trick as implemented in [7]. To ensure a fair comparison, our LE-PDE use temporal bundling of $S = 25$ time steps as in MP-PDE and FNO-PF. We perform hyperparameter search over latent dimension of $\{64, 128\}$ and use the model with best validation performance. In addition, we compare with downsampled ground-truth (WENO5), which uses a classical 5th-order WENO scheme [65] and explicit Runge-Kutta 4 solver [66, 67] to generate the ground-truth data and downsampled to the specified resolution. For all models, we autoregressively roll out to predict the states starting at step 50 until step 250, and record the accumulated MSE, runtime and representation dimension (the dimension of state to update at each time step). Details of the experiments are given in Appendix D.

Results. The result is shown in Table 1. We see that since LE-PDE uses 7.8 to 39-fold smaller representation dimension, it achieves significant smaller runtime compared to the MP-PDE model

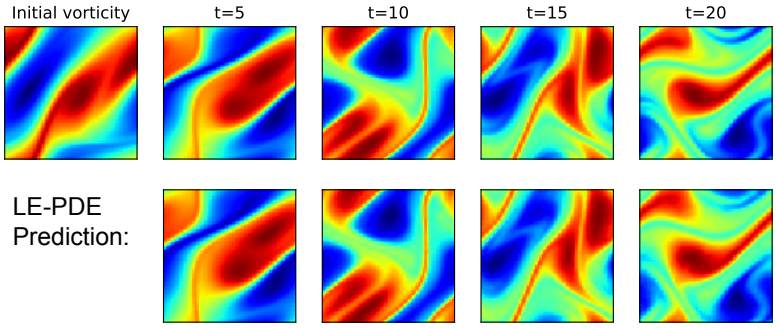


Figure 2: Visualization of rollout for 2D Navier-Stokes PDE ($Re = 10^4$), for ground-truth (upper) and LE-PDE (lower, trained with $\nu = 10^{-4}, N = 10^4$). LE-PDE captures detailed dynamics faithfully.

(which is much faster than the classical WENO5 scheme). Here we record the latent evolution time (LE-PDE evo) which is the total time for 200-step latent evolution, and the full time (LE-PDE full), which also includes decoding to the input space at each time step. The time for "LE-PDE evo" is relevant when the downstream application is only concerned with state at long-term future (*e.g.* [4]); the time for "LE-PDE full" is relevant when every intermediate prediction is also important. LE-PDE achieves up to 15× speed-up with "LE-PDE evo" and 4× speed-up with "LE-PDE full".

With above 7.8× compression and above 4× speed-up, LE-PDE still achieves competitive accuracy. For **E1** scenario, it significantly outperforms both original versions of FNO-RNN and MP-PDE, and only worse than the improved version of FNO-PF. For **E3**, LE-PDE outperforms both versions of FNO-RNN and FNO-PF, and the performance is on par with MP-PDE and sometimes better. For **E2**, LE-PDE outperforms all state-of-the-art models by a large margin. Fig. 4 in Appendix D shows our model's representative rollout compared to ground-truth. We see that during long-rollout, our model captures the shock formation faithfully. This 1D benchmark shows that LE-PDE is able to achieve significant speed-up, generalize to novel PDE parameters and achieve competitive long-term rollout.

4.2 2D Navier-Stokes flow

Data and Experiments. We test LE-PDE in a 2D benchmark [14] of Navier-Stokes equation. Navier-Stokes equation has wide application science and engineering, including weather forecasting, jet engine design, etc. It becomes more challenging to simulate when entering the turbulent phase, which shows multiscale dynamics and chaotic behavior. Specifically, we test our model in a viscous, incompressible fluid in vorticity form in a unit torus:

$$\partial_t w(t,x) + u(t,x) \cdot \nabla w(t,x) = \nu \Delta w(t,x) + f(x), \quad x \in (0,1)^2, t \in (0,T] \tag{10}$$

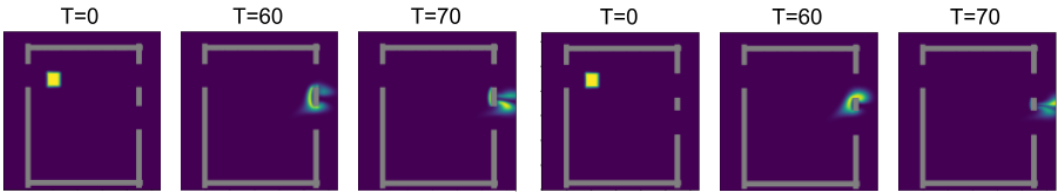
$$\nabla \cdot u(t,x) = 0, \quad x \in (0,1)^2, t \in [0,T] \tag{11}$$

$$w(0,x) = w_0(x), \quad x \in (0,1)^2 \tag{12}$$

Here $w(t,x) = \nabla \times u(t,x)$ is the vorticity, $\nu \in \mathbb{R}_+$ is the viscosity coefficient. The domain is discretized into 64×64 grid. We test with viscosities of $\nu = 10^{-3}, 10^{-4}, 10^{-5}$. The fluid is turbulent for $\nu = 10^{-4}, 10^{-5}$ ($Re \geq 10^4$). We compare state-of-the-art learning-based model Fourier Neural Operator (FNO) [14] for this problem, and strong baselines of TF-Net [26], U-Net [68] and ResNet [69]. For FNO, the FNO-2D performs autoregressive rollout, and FNO-3D directly maps the past 10 steps into all future steps. To ensure a fair comparison, here our LE-PDE uses past 10 steps to predict

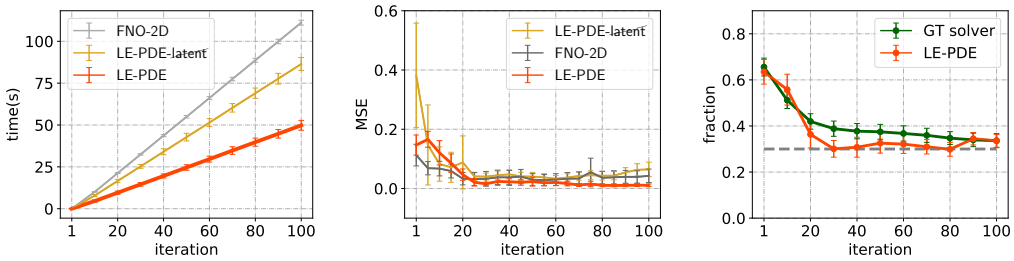
Table 2: Performance of different models in 2D Navier-Stokes flow. Runtime is using the $\nu = 10^{-3}, N = 1000$ for predicting 40 steps in the future.

Method	Representation dimensions	Runtime full $[ms]$	Runtime evo $[ms]$	$\nu = 10^{-3}$ $T = 50$ $N = 1000$	$\nu = 10^{-4}$ $T = 30$ $N = 1000$	$\nu = 10^{-4}$ $T = 30$ $N = 10000$	$\nu = 10^{-5}$ $T = 20$ $N = 1000$
FNO-3D [14]	4096	24	24	0.0086	0.1918	0.0820	0.1893
FNO-2D [14]	4096	140	140	0.0128	0.1559	0.0834	0.1556
U-Net [68]	4096	813	813	0.0245	0.2051	0.1190	0.1982
TF-Net [26]	4096	428	428	0.0225	0.2253	0.1168	0.2268
ResNet [69]	4096	317	317	0.0701	0.2871	0.2311	0.2753
LE-PDE (ours)	256	48	15	0.0146	0.1936	0.1115	0.1862



(a) Trajectory generated by ground-truth solver with an initial randomly generated boundary parameter.

(b) Trajectory generated by ground-truth solver with boundary parameter optimized by LE-PDE.



(c) Runtime.

(d) Objective function.

(e) Smoke amount at lower outlet.

Figure 3: Numerical results associated with inverse optimization of boundary (inlet and outlet designing) in Sec. 4.3. (a) shows a trajectory generated by ground-truth (GT) solver with an initial randomly generated boundary parameters (y-position of inlet and two outlets), with lower outlet passing 55.18% of smoke; (b) with optimized boundary parameters, with lower outlet passing 31.79% of smoke, very near the objective percentage of 30%. (c) Runtime and (d) learning curve (Eq. 7) for inverse optimization at different iteration steps; (e) For LE-PDE, fraction of smoke passing through the lower outlet computed by GT solver (green) and estimated by LE-PDE (orange). Error bar denotes 95% confidence interval over 50 runs with random initial conditions.

one future step and temporal bundling $S = 1$ (no bundling), the same setting as in FNO-2D. We use relative L2 norm (normalized by ground-truth's L2 norm) as metric, same as in [14].

Results. The results are shown in Table 2. Similar to 1D case, LE-PDE is able to compress the representation dimension by 16-fold. Hence, compared with FNO-2D which is also autoregressive, LE-PDE achieves 9.3-fold speed-up with latent evolution and 2.9-fold speed-up with full decoding. Compared with FNO-3D that directly maps all input time steps to all output times steps (which cannot generalize beyond the time range given), LE-PDE's runtime is still 1.6× faster for latent evolution. For rollout L2 loss, LE-PDE significantly outperforms strong baselines of ResNet and U-Net, and TF-Net which is designed to model turbulent flow. Its performance is on par with FNO-3D with $\nu = 10^{-4}, N = 1000$ and the most difficult $\nu = 10^{-5}, N = 1000$ and slightly underperforms FNO-2D in other scenarios. Fig. 2 shows the visualization of LE-PDE comparing with ground-truth, under the turbulent $\nu = 10^{-4}, N = 10000$ scenario. We see that LE-PDE captures the detailed dynamics accurately. For more details, see Appendix E. To explore how LE-PDE can model and accelerate the simulation of systems with a larger scale, in Appendix F we explore modeling a 3D Navier-Stokes flow with millions of cells per time step, and show more significant speed-up.

4.3 Accelerating inverse optimization of boundary conditions

Data and Experiments. In this subsection, we set out to answer question (3), *i.e.* Can LE-PDE improve and speed up inverse optimization? We are interested in long time frame scenarios where the pre-defined objective L_d in Eq. (7) depends on the prediction after long-term rollout. Such problems are challenging and have implications in engineering, *e.g.* fluid control [70, 71], laser design for laser-plasma interaction [4] and nuclear fusion [72]. To evaluate, we build a 2D Navier-Stokes flow in a family of boundary conditions using PhiFlow [73] as our ground-truth solver, shown in Fig. 3a, 3b. Specifically, we create a cubical boundary with one inlet and two outlets on a grid space of size 128^2. We initialize the velocity and smoke on this domain and advect the dynamics by performing rollout. The objective

Table 3: Comparison of LE-PDE with baselines.

	GT-solver Error (Model estimated Error)	Runtime $[s]$
LE-PDE-~~latent~~	0.305 (0.123)	86.42
FNO-2D	0.124 (0.004)	111.14
LE-PDE (ours)	**0.035** (0.036)	**49.81**

of the inverse design here is to optimize the boundary parameter p, *i.e.* the y-locations of the inlet and outlets, so that the amount of smoke passing through the two outlets coincides with pre-specified proportions 0.3 and 0.7. This setting is challenging since a slight change in boundary (up to a few cells) can have large influence in long-term rollout and the predefined objective.

As baseline methods, we use our LE-PDE's ablated version without latent evolution (essentially a CNN, which we call LE-PDE-~~latent~~) and the FNO-2D [14], both of which update the states in input space, while LE-PDE evolves in a 128-dimensional latent space ($128\times$ compression). To ensure a fair comparison, all models predict the next step using 1 past step without temporal bundling, and trained with 4-step rollout. We train all models with 400 generated trajectories of length 100 and test with 40 trajectories. After training, we perform inverse optimization w.r.t. the boundary parameter p with the trained models using Eq. 7, starting with 50 initial configurations each with random initial location of smoke and random initial configuration of p. For LE-PDE-~~latent~~ and FNO-2D, they need to backpropagate through 80 steps of rollout in input space as in [45, 74], while LE-PDE backpropagates through 80 steps of latent rollout. Then the optimized boundary parameter is fed to the ground-truth solver for rollout and evaluate. For the optimized parameter, we measure the total amount of smoke simulated by the solver passing through two respective outlets and take their ratio. The evaluation metric is the average ratio across all 50 configurations: see also Appendix G.

Results. We observe that LE-PDE improves the overall speed by 73% compared with LE-PDE-~~latent~~ and by 123% compared with FNO-2D (Fig. 3c, Table 3). The result indicates a corollary of the use of low dimensional representation because Jacobian matrix of evolution operator is reduced to be of smaller size and suppresses the complexity associated with the chain rule to compute gradients of the objective function. While achieving the significant speed-up, the capability of the LE-PDE to design the boundary is also reasonable. Fig. 3d shows the loss of the objective function achieved the lowest value while the others are comparably large. The estimated proportion of smoke hit the target fraction 0.3 at an early stage of design iterations and coincide with the fraction simulated by the ground-truth solver in the end (Fig. 3e). As Table 3 shows, FNO-2D achieves the lowest score in model estimated error from the target fraction 0.3 while its ground-truth solver (GT-solver) error is $30\times$ larger. This shows "overfitting" of the boundary parameter by FNO-2D, *i.e.* the optimized parameter is not sufficiently generalized to work for a ground-truth solver. In this sense, LE-PDE achieved to design the most generalized boundary parameter: the difference between the two errors is the smallest among the others.

4.4 Ablation study

In this section, we investigate how each component of our LE-PDE influences the performance. Importantly, we are interested in how each of the three components: multi-step loss $L_{\text{multi-step}}$, latent consistency loss $L_{\text{consistency}}$ and reconstruction loss L_{recons} contribute to the performance, and how the time horizon M and the latent dimension d_z influence the result. For dataset, we focus on representative scenarios in 1D (Sec. 4.1) and 2D (Sec. 4.2), specifically the **E2** scenario with $(n_t, n_x) = (250, 50)$ for 1D, and $(\nu = 10^{-5}, T = 20, N = 1000)$ scenario for 2D, which lies at mid- to difficult spectrum of each dataset. We have observed similar trends in other scenarios. From Table 4, we see that all three components $L_{\text{multi-step}}$, $L_{\text{consistency}}$ and L_{recons} are necessary and pivotal in ensuring a good performance. The time horizon M in the loss is also important. If too short (*e.g.* $M = 1$), it does not encourage accurate long-term rollout. Increasing M helps reducing error, but will be countered by less number of examples (since having to leave room for more steps in the future). We find the sweet spot is at $M = 4$, which achieves a good tradeoff. In Fig. 6 in Appendix H, we show how the error and evolution runtime change with varying size of latent dimension d_z. We observe that reduction of runtime with decreasing latent dimension d_z, and that the error is lowest at $d_z = 64$ for 1D and $d_z = 256$ for 2D, suggesting the intrinsic dimension of each problem.

Table 4: Error for ablated versions of LE-PDE in 1D and 2D.

	1D	2D
LE-PDE (ours)	**1.127**	**0.1861**
no $L_{\text{multi-step}}$	3.337	0.2156
no $L_{\text{consistency}}$	6.386	0.2316
no L_{recons}	1.506	0.2025
Time horizon $M = 1$	5.710	0.2860
Time horizon $M = 3$	1.234	0.2010
Time horizon $M = 4$	1.127	0.1861
Time horizon $M = 6$	1.924	0.1923

5 Discussion and Conclusion

In this work, we have introduced LE-PDE, a simple, fast and scalable method for accelerating simulation and inverse optimization of PDEs, including its simple architecture, objective and inverse optimization techniques. Compared with state-of-the-art deep learning-based surrogate models, we demonstrate that it achieves up to 128 $\times$ reduction in the dimensions to update and up to 15$\times$ improvement in speed, while achieving competitive accuracy. Ablation study shows both multi-step objective and latent-consistency objectives are pivotal in ensuring accurate long-term rollout. We hope our method will make a useful step in accelerating simulation and inverse optimization of PDEs, pivotal in science and engineering.

Acknowledgments and Disclosure of Funding

We thank Sophia Kivelson, Jacqueline Yau, Rex Ying, Paulo Alves, Frederico Fiuza, Jason Chou, Qingqing Zhao for discussions and for providing feedback on our manuscript. We also gratefully acknowledge the support of DARPA under Nos. HR00112190039 (TAMI), N660011924033 (MCS); ARO under Nos. W911NF-16-1-0342 (MURI), W911NF-16-1-0171 (DURIP); NSF under Nos. OAC-1835598 (CINES), OAC-1934578 (HDR), CCF-1918940 (Expeditions), NIH under No. 3U54HG010426-04S1 (HuBMAP), Stanford Data Science Initiative, Wu Tsai Neurosciences Institute, Amazon, Docomo, GSK, Hitachi, Intel, JPMorgan Chase, Juniper Networks, KDDI, NEC, and Toshiba.

The content is solely the responsibility of the authors and does not necessarily represent the official views of the funding entities.

References

[1] P. Lynch, "The origins of computer weather prediction and climate modeling," *Journal of computational physics*, vol. 227, no. 7, pp. 3431–3444, 2008.

[2] M. Athanasopoulos, H. Ugail, and G. G. Castro, "Parametric design of aircraft geometry using partial differential equations," *Advances in Engineering Software*, vol. 40, no. 7, pp. 479–486, 2009.

[3] F. Carpanese, "Development of free-boundary equilibrium and transport solvers for simulation and real-time interpretation of tokamak experiments," EPFL, Tech. Rep., 2021.

[4] N. Sircombe, T. Arber, and R. Dendy, "Kinetic effects in laser-plasma coupling: Vlasov theory and computations," in *Journal de Physique IV (Proceedings)*, vol. 133. EDP sciences, 2006, pp. 277–281.

[5] R. Courant, K. Friedrichs, and H. Lewy, "On the partial difference equations of mathematical physics," *IBM journal of Research and Development*, vol. 11, no. 2, pp. 215–234, 1967.

[6] T. Lelievre and G. Stoltz, "Partial differential equations and stochastic methods in molecular dynamics," *Acta Numerica*, vol. 25, pp. 681–880, 2016.

[7] J. Brandstetter, D. E. Worrall, and M. Welling, "Message passing neural PDE solvers," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=vSix3HPYKSU

[8] D. E. Keyes, D. R. Reynolds, and C. S. Woodward, "Implicit solvers for large-scale nonlinear problems," in *Journal of Physics: Conference Series*, vol. 46, no. 1. IOP Publishing, 2006, p. 060.

[9] Y. Dubois and R. Teyssier, "Cosmological MHD simulation of a cooling flow cluster," *Astronomy & Astrophysics*, vol. 482, no. 2, pp. L13–L16, 2008.

[10] P. Chatelain, A. Curioni, M. Bergdorf, D. Rossinelli, W. Andreoni, and P. Koumoutsakos, "Billion vortex particle direct numerical simulations of aircraft wakes," *Computer Methods in Applied Mechanics and Engineering*, vol. 197, no. 13-16, pp. 1296–1304, 2008.

[11] L. T. Biegler, O. Ghattas, M. Heinkenschloss, and B. v. Bloemen Waanders, "Large-scale pde-constrained optimization: an introduction," in *Large-Scale PDE-Constrained Optimization*. Springer, 2003, pp. 3–13.

[12] K. Um, R. Brand, Y. R. Fei, P. Holl, and N. Thuerey, "Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6111–6122, 2020.

[13] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. Battaglia, "Learning to simulate complex physics with graph networks," in *International Conference on Machine Learning*. PMLR, 2020, pp. 8459–8468.

[14] Z. Li, N. B. Kovachki, K. Azizzadenesheli, B. liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, "Fourier neural operator for parametric partial differential equations," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=c8P9NQVtmnO

[15] M. Tang, Y. Liu, and L. J. Durlofsky, "A deep-learning-based surrogate model for data assimilation in dynamic subsurface flow problems," *Journal of Computational Physics*, vol. 413, p. 109456, 2020.

[16] T. Wu, Q. Wang, Y. Zhang, R. Ying, K. Cao, R. Sosic, R. Jalali, H. Hamam, M. Maucec, and J. Leskovec, "Learning large-scale subsurface simulations with a hybrid graph network simulator," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 4184–4194.

[17] D. Kochkov, J. A. Smith, A. Alieva, Q. Wang, M. P. Brenner, and S. Hoyer, "Machine learning–accelerated computational fluid dynamics," *Proceedings of the National Academy of Sciences*, vol. 118, no. 21, 2021.

[18] A. Sanchez, D. Kochkov, J. A. Smith, M. Brenner, P. Battaglia, and T. J. Pfaff, "Learning latent field dynamics of PDEs," *Advances in Neural Information Processing Systems*, 2020.

[19] N. Watters, D. Zoran, T. Weber, P. Battaglia, R. Pascanu, and A. Tacchetti, "Visual interaction networks: Learning a physics simulator from video," *Advances in neural information processing systems*, vol. 30, 2017.

[20] S. van Steenkiste, M. Chang, K. Greff, and J. Schmidhuber, "Relational neural expectation maximization: Unsupervised discovery of objects and their interactions," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=ryH20GbRW

[21] S.-M. Udrescu and M. Tegmark, "Symbolic pregression: discovering physical laws from distorted video," *Physical Review E*, vol. 103, no. 4, p. 043307, 2021.

[22] C. Gelada, S. Kumar, J. Buckman, O. Nachum, and M. G. Bellemare, "Deepmdp: Learning continuous latent space models for representation learning," in *International Conference on Machine Learning*. PMLR, 2019, pp. 2170–2179.

[23] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, "Learning latent dynamics for planning from pixels," in *International conference on machine learning*. PMLR, 2019, pp. 2555–2565.

[24] R. C. Julian, E. Heiden, Z. He, H. Zhang, S. Schaal, J. J. Lim, G. S. Sukhatme, and K. Hausman, "Scaling simulation-to-real transfer by learning a latent space of robot skills," *The International Journal of Robotics Research*, vol. 39, no. 10-11, pp. 1259–1278, 2020.

[25] A. X. Lee, A. Nagabandi, P. Abbeel, and S. Levine, "Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model," *Advances in Neural Information Processing Systems*, vol. 33, pp. 741–752, 2020.

[26] R. Wang, K. Kashinath, M. Mustafa, A. Albert, and R. Yu, "Towards physics-informed deep learning for turbulent flow prediction," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 1457–1466.

[27] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. W. Battaglia, "Learning mesh-based simulation with graph networks," in *International Conference on Learning Representations*, 2021.

[28] Z. Li and A. B. Farimani, "Graph neural network-accelerated lagrangian fluid simulation," *Computers & Graphics*, vol. 103, pp. 201–211, 2022.

[29] Ö. Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger, "3d u-net: learning dense volumetric segmentation from sparse annotation," in *International conference on medical image computing and computer-assisted intervention*. Springer, 2016, pp. 424–432.

[30] M. Raissi, "Deep hidden physics models: Deep learning of nonlinear partial differential equations," *The Journal of Machine Learning Research*, vol. 19, no. 1, pp. 932–955, 2018.

[31] Y. Zhu and N. Zabaras, "Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification," *Journal of Computational Physics*, vol. 366, pp. 415–447, 2018.

[32] S. Bhatnagar, Y. Afshar, S. Pan, K. Duraisamy, and S. Kaushik, "Prediction of aerodynamic flow fields using convolutional neural networks," *Computational Mechanics*, vol. 64, no. 2, pp. 525–545, 2019.

[33] Y. Khoo, J. Lu, and L. Ying, "Solving parametric pde problems with artificial neural networks," *European Journal of Applied Mathematics*, vol. 32, no. 3, pp. 421–435, 2021.

[34] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, "Neural operator: Graph kernel network for partial differential equations," *arXiv preprint arXiv:2003.03485*, 2020.

[35] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, A. Stuart, K. Bhattacharya, and A. Anandkumar, "Multipole graph neural operator for parametric partial differential equations," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6755–6766, 2020.

[36] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. Karniadakis, "Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. nature mach. intell. 3 (3), 218–229 (2021)."

[37] K. T. Butler, J. M. Frost, J. M. Skelton, K. L. Svane, and A. Walsh, "Computational materials design of crystalline solids," *Chemical Society Reviews*, vol. 45, no. 22, pp. 6138–6146, 2016.

[38] I. Vernon, M. Goldstein, and R. Bower, "Galaxy formation: Bayesian history matching for the observable universe," *Statistical science*, pp. 81–90, 2014.

[39] D. Williamson, M. Goldstein, L. Allison, A. Blaker, P. Challenor, L. Jackson, and K. Yamazaki, "History matching for exploring and reducing climate model parameter space using observations and a large perturbed physics ensemble," *Climate dynamics*, vol. 41, no. 7, pp. 1703–1729, 2013.

[40] D. S. Oliver and Y. Chen, "Recent progress on reservoir history matching: a review," *Computational Geosciences*, vol. 15, no. 1, pp. 185–221, 2011.

[41] O. Talagrand and P. Courtier, "Variational assimilation of meteorological observations with the adjoint vorticity equation. i: Theory," *Quarterly Journal of the Royal Meteorological Society*, vol. 113, no. 478, pp. 1311–1328, 1987.

[42] J. Tromp, C. Tape, and Q. Liu, "Seismic tomography, adjoint methods, time reversal and banana-doughnut kernels," *Geophysical Journal International*, vol. 160, no. 1, pp. 195–216, 2005.

[43] H. B. Keller, *Numerical solution of two point boundary value problems*. SIAM, 1976.

[44] J. T. Betts, "Survey of numerical methods for trajectory optimization," *Journal of guidance, control, and dynamics*, vol. 21, no. 2, pp. 193–207, 1998.

[45] K. R. Allen, T. Lopez-Guevara, K. Stachenfeld, A. Sanchez-Gonzalez, P. Battaglia, J. Hamrick, and T. Pfaff, "Physical design using differentiable learned simulators," *arXiv preprint arXiv:2202.00728*, 2022.

[46] R. Y. Rubinstein and D. P. Kroese, "The cross-entropy method: A unified approach to monte carlo simulation, randomized optimization and machine learning," *Information Science & Statistics, Springer Verlag, NY*, 2004.

[47] A. Treuille, A. Lewis, and Z. Popović, "Model reduction for real-time fluids," *ACM Transactions on Graphics (TOG)*, vol. 25, no. 3, pp. 826–834, 2006.

[48] G. Berkooz, P. Holmes, and J. L. Lumley, "The proper orthogonal decomposition in the analysis of turbulent flows," *Annual review of fluid mechanics*, vol. 25, no. 1, pp. 539–575, 1993.

[49] M. Gupta and S. G. Narasimhan, "Legendre fluids: a unified framework for analytic reduced space modeling and rendering of participating media," in *Symposium on Computer Animation*, 2007, pp. 17–25.

[50] M. Wicke, M. Stanton, and A. Treuille, "Modular bases for fluid dynamics," *ACM Transactions on Graphics (TOG)*, vol. 28, no. 3, pp. 1–8, 2009.

[51] B. Long and E. Reinhard, "Real-time fluid simulation using discrete sine/cosine transforms," in *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, 2009, pp. 99–106.

[52] T. De Witt, C. Lessig, and E. Fiume, "Fluid simulation using laplacian eigenfunctions," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 1, pp. 1–11, 2012.

[53] T. Kim and J. Delaney, "Subspace fluid re-simulation," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 4, pp. 1–9, 2013.

[54] B. Liu, G. Mason, J. Hodgson, Y. Tong, and M. Desbrun, "Model-reduced variational fluid simulation," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 6, pp. 1–12, 2015.

[55] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," in *4th International Conference on Learning Representations*, 2016. [Online]. Available: https://arxiv.org/abs/1511.06434

[56] S. Wiewel, M. Becher, and N. Thuerey, "Latent space physics: Towards learning the temporal evolution of fluid flow," in *Computer graphics forum*, vol. 38, no. 2. Wiley Online Library, 2019, pp. 71–82.

[57] B. Kim, V. C. Azevedo, N. Thuerey, T. Kim, M. Gross, and B. Solenthaler, "Deep fluids: A generative network for parameterized fluid simulations," in *Computer Graphics Forum*, vol. 38, no. 2. Wiley Online Library, 2019, pp. 59–70.

[58] K. Lee and K. T. Carlberg, "Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders," *Journal of Computational Physics*, vol. 404, p. 108973, 2020.

[59] S. Wiewel, B. Kim, V. C. Azevedo, B. Solenthaler, and N. Thuerey, "Latent space subdivision: stable and controllable time predictions for fluid flow," in *Computer Graphics Forum*, vol. 39, no. 8. Wiley Online Library, 2020, pp. 15–25.

[60] P. R. Vlachas, G. Arampatzis, C. Uhler, and P. Koumoutsakos, "Multiscale simulations of complex systems by learning their effective dynamics," *Nature Machine Intelligence*, vol. 4, no. 4, pp. 359–366, 2022.

[61] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (Poster)*, 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[62] N. Tishby, F. C. Pereira, and W. Bialek, "The information bottleneck method," *arXiv preprint physics/0004057*, 2000.

[63] T. Wu and I. Fischer, "Phase transitions for the information bottleneck in representation learning," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=HJloElBYvB

[64] Y. Bar-Sinai, S. Hoyer, J. Hickey, and M. P. Brenner, "Learning data-driven discretizations for partial differential equations," *Proceedings of the National Academy of Sciences*, vol. 116, no. 31, pp. 15 344–15 349, 2019.

[65] C.-W. Shu, "High-order finite difference and finite volume weno schemes and discontinuous galerkin methods for cfd," *International Journal of Computational Fluid Dynamics*, vol. 17, no. 2, pp. 107–118, 2003.

[66] C. Runge, "Über die numerische auflösung von differentialgleichungen," *Mathematische Annalen*, vol. 46, no. 2, pp. 167–178, 1895.

[67] W. Kutta, "Beitrag zur naherungsweisen integration totaler differentialgleichungen," *Z. Math. Phys.*, vol. 46, pp. 435–453, 1901.

[68] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[69] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[70] A. McNamara, A. Treuille, Z. Popović, and J. Stam, "Fluid control using the adjoint method," *ACM Transactions On Graphics (TOG)*, vol. 23, no. 3, pp. 449–456, 2004.

[71] P. Holl, N. Thuerey, and V. Koltun, "Learning to control pdes with differentiable physics," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=HyeSin4FPB

[72] A. Zylstra, O. Hurricane, D. Callahan, A. Kritcher, J. Ralph, H. Robey, J. Ross, C. Young, K. Baker, D. Casey *et al.*, "Burning plasma achieved in inertial fusion," *Nature*, vol. 601, no. 7894, pp. 542–548, 2022.

[73] "Phiflow," https://github.com/tum-pbs/PhiFlow.

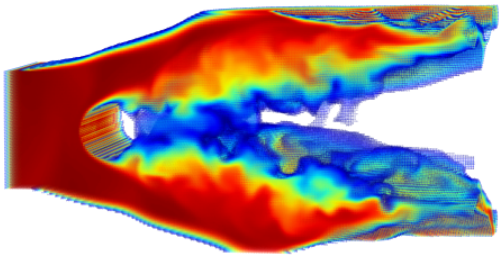
[74] Q. Zhao, D. B. Lindell, and G. Wetzstein, "Learning to solve pde-constrained inverse problems with graph networks," *International Conference on Machine Learning*, 2022.

[75] K. Roberts and H. L. Berk, "Nonlinear evolution of a two-stream instability," *Physical Review Letters*, vol. 19, no. 6, p. 297, 1967.

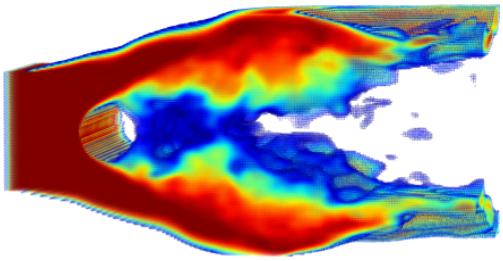
Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] In Appendix C.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] In Appendix I
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] The Appendix includes full details on model architecture, training and evaluation to reproduce the experimental results. Code and data will be released upon publication of the paper.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Important training details are included in main text, and full details to reproduce the experiments are included in the Appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Yes in Section 4.4. For Sections 4.1 and 4.2, the benchmarks do not include the error bars and we use the same format.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] In Appendix D,E,F,G.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] In Appendix D,E,F,G.
   (b) Did you mention the license of the assets? [Yes] In Appendix D,E,F,G.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] In Appendix D,E,F,G.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes] In Appendix D,E,F,G.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes] In Appendix D,E,F,G.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

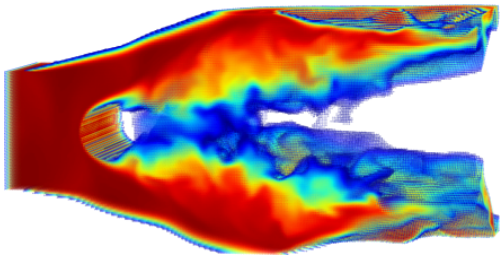
Appendix



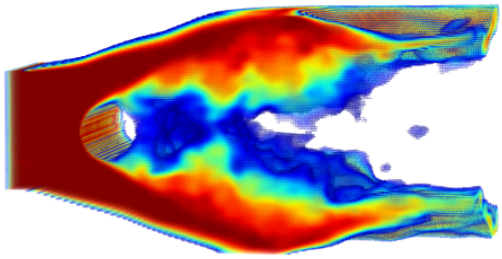
(a) Initial condition at $t = 0$



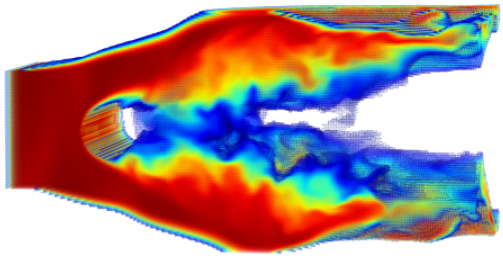
(b) LE-PDE prediction at $t = 20$



(c) Ground-truth at $t = 20$



(d) LE-PDE prediction at $t = 40$



(e) Ground-truth at $t = 40$

Figure S1: Visualization of LE-PDE testing on predicting the dynamics of turbulent 3D Navier-Stokes flow through the cylinder with a novel Reynolds number (detail in Appendix F). The input domain of size $2 \times 1 \times 1$ is discretized into a 3D grid of $256 \times 128 \times 128$, resulting in 4.19 million cells per time step. **Compression:** LE-PDE learns latent dynamics with latent dimension of $d_z = 128$, achieving a **130,000**$\times$ reduction in representation dimension, compared with 4.19 million cells times 4 features per cell (ρ, v_x, v_y, v_z) in input space. **Prediction quality:** The visualization is shown at a cross-section of $x = 50/128 \times 1$ along the direction of the cylinder. We see that compared with ground-truth (c)(e), LE-PDE (b)(d) captures the turbulent dynamics reasonably well, predicting both high-level and low-level dynamics in a qualitatively faithful way. This shows the scalability of LE-PDE to large-scale simulations of PDE. **Speed-up:** To predict the state at $t = 40$, on an Nvidia Quadro RTX 8000 48GB GPU, the ground-truth solver PhiFlow [73] uses 70.80s, an ablation our LE-PDE-~~latent~~ without latent evolution (essentially a CNN) takes 1.03s, while our LE-PDE takes only 0.084s. LE-PDE achieves an **840**$\times$ speed-up compared to the ground-truth solver, and **12.3**$\times$ speed-up compared to the ablation model without latent evolution.

In the Appendix, we provide details that complement the main text. In Appendix A, we give a brief introduction to classical solvers. In Appendix B, we explain details about boundary interpolation and annealing technique used in Section 4.3. In Appendix C, we give full explanation on the architecture of LE-PDE used throughout the experiments. The following three sections explain details on parameter settings of experiments: 1D family of nonlinear PDEs (Appendix D), 2D Navier-Stokes flow (Appendix E) and 3D Navier-Stokes flow (Appendix F). In appendix G, we give details of boundary inverse optimization conducted in Section 4.3. In appendix H, we show ablation study for LE-PDE's important parameters. In Appendix I, we discuss the broader social impact of our work. In appendix J, we give comparison of trade-off between some metrics for LE-PDE and some

strong baselines. In addition, in Appendix K, we compare LE-PDE to another model exploiting latent evolution method from several aspects. We present the influence of varying noise amplitude with some tables in Appendix L. Finally, in Appendix M, we show the ablation study for various encoders in different scenarios.

A Classical Numerical Solvers for PDEs

We refer the readers to [7] Section 2.2 and Appendix for a high-level introduction of the classical PDE solvers. One thing that is in common with the Finite Difference Method (FDM), Finite Volume Method (FVM) is that they all need to update the state of each cell at each time step. This stems from that the methods require discretization of the domain $\mathbb{X}$ and solution $\mathbf{u}$ into a grid X. For large-systems with millions or billions of cells, it will result in extremely slow simulation, as is also shown in Appendix F where a classical solver takes extremely long to evolve a 3D system with millions of cells per time step.

B Boundary Interpolation and Annealing Technique

Boundary interpolation. In order to allow gradients to pass through to the boundary parameter p, we introduce a *continuous boundary mask* that continuously interpolates a discrete boundary mask and continuous variables. Here, for the later convenience, we regard a mask as a function from a grid structure $\mathbb{N}^{\times 2}_{\leq 128}$ to $[0, 1]$. Because boundary is composed by 1-dimensional segments, we use a 1-dimensional sigmoid function for the interpolation. Specifically, we define a sigmoid-interpolation function on a segment as a map to a real from a natural number i conditioned by a pair of continuous variables x_1, x_2 and positive real β:

$$f(\ i \mid x_1, x_2, \beta) = \begin{cases} \mathrm{sigmoid}(\frac{i-x_1}{\beta}), & i \leq x_1, \\ \mathrm{sigmoid}(\frac{x_2-i}{\beta}), & x_2 \leq i, \\ \mathrm{sigmoid}(\frac{GM_{-1}(|i-x_1|,|i-x_2|)}{\beta}), & x_1 < i < x_2. \end{cases} \tag{13}$$

Here, x_1 and x_2 are the location of the edge of the line-segment boundary, which is to be optimized during inverse optimization. $GM_{-1}(|i-x_1|, |i-x_2|) = (\frac{1}{2}(|i-x_1|^{-1} + |i-x_2|^{-1}))^{-1}$ denotes the harmonic mean[2], which is influenced more by the smaller of $|i-x_1|$ and $|i-x_2|$, so it is a *soft* version of the distance to the nearest edge inside the line segment of $x_1 < i < x_2$. When β tends to 0, the function f converges to a binary valued function: see also Fig. S2.

We define a continuous boundary function CB on a segment in a grid to be the pullback of the sigmoid-interpolation function with the projection to 1-dimensional discretized line (*i.e.*, take a projection of the pair of integers onto a 1-dimensional segment and apply f):

$$CB(\ (i,j) \mid (x_1, x_2), \beta) = \begin{cases} f(\ i \mid (x_1, x_2), \beta), & \text{if } (i,j) \text{ is in a horizontal segment,} \\ f(\ j \mid (x_1, x_2), \beta), & \text{if } (i,j) \text{ is in a vertical segment.} \end{cases} \tag{14}$$

Finally, a continuous boundary mask on a grid is obtained by (tranformation by a function $1-x$ and) taking the maximum on a set of CBs on boundary segments on the grid (see also Fig. S3). The boundary interpolation allows the gradient to pass through the boundary mask and able to optimize the location of the edge of line segments (*e.g.* x_1, x_2).

Boundary annealing. As we see above, β can be seen as a temperature hyperparameter, and the smaller it is, the more the boundary mask approximates a binary valued mask, and the less cells the boundary directly influences. At the beginning of the optimization, the parameter of the boundary (locations x_1, x_2 of each line segment) may be far away from the optimal location. Having a small temperature β would result in vanishing gradient for the optimization, and very sparse interaction where the boundary mainly interact with its immediate neighbors, resulting in that very small gradient signal to optimize. Therefore, we introduce an annealing technique for the boundary optimization,

[2] $GM_\gamma(x,y) = (\frac{1}{2}(x^\gamma + y^\gamma))^{1/\gamma}$ is generalized mean with order γ. The harmonic mean $GM_{-1}(x,y)$ interpolates between arithmetic mean $GM_1(x,y) = \frac{1}{2}(x+y)$ and the minimum $GM_{-\infty}(x,y) = \min(x,y)$, and is influenced more by the smaller of x and y.

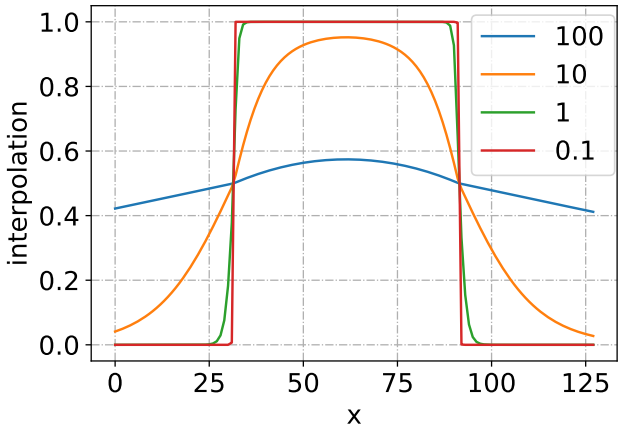


Figure S2: The interpolation of binary valued function by a sigmoid-interpolation function. Continuous variables (x_1, x_2) are set to be $(31.5, 91.3)$. The continuous variables define edges of a continuous segment.

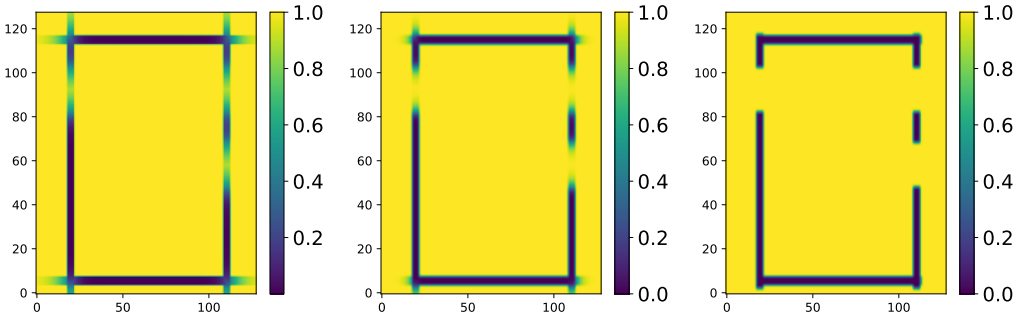


Figure S3: Continuous bounds with different parameters $\beta = 5$ (left), 2 (middle) and 0.01 (right). As β decreases, the edges of the boundaries tend to have steeper slopes.

where at the beginning, we start at a larger β_0, and linearly tune it down until at the end reaching a much smaller β. The larger β at the beginning allows denser gradient at the beginning of inverse optimization, where the location of the boundary can also influence more cells, producing more gradient signals. The smaller β at the end allows more accurate boundary location optimization at the end, where we want to reduce the bias introduced by the boundary interpolation.

C Model Architecture for LE-PDE

Here we detail the architecture of LE-PDE, complementary to Sec. 3.1. This architecture is used throughout all experiment, with just a few hyperparameter (*e.g.* latent dimension d_z, number of convolution layers) depending on the dimension (1D, 2D, 3D) of the problem. We first detail the 4 architectural components of LE-PDE, and then discuss its current limitations.

Dynamic encoder q. The dynamic encoder q consists of one CNN layer with (kernel-size, stride, padding) = $(3, 1, 1)$ and ELU activation, followed by F_q convolution blocks, then followed by a flatten operation and an MLP with 1 layer and linear activation that outputs a d_z-dimensional vector $\mathbf{z}^k \in \mathbb{R}^{d_z}$ at time step k. Each of the F_q convolution block consists of a convolution layer with (kernel-size, stride, padding) = $(4, 2, 1)$ followed by group normalization [74] (number of groups=2) and ELU activation [75]. The channel size of each convolution block follows the standard exponentially increasing pattern, *i.e.* the first convolution block has C channels, the second has $C \times 2^1$ channels, ...

the nth convolution block has $C \times 2^{n-1}$ channels. The larger channel size partly compensates for smaller spatial dimensions of feature map for higher layers.

Static encoder r. For the static encoder r, depending on the static parameter p, it can be an F_r-layer MLP (as in 1D experiment Sec. 4.1 and 3D experiment Appendix F), or a similar CNN+MLP architecture as the dynamic encoder (as in Sec. 4.3 that takes as input the boundary mask). If using MLP, it uses F_r layers with ELU activation and the last layer has linear activation. In our experiments, we select $F_r \in \{0, 1, 2\}$, and when $F_r = 0$, it means no layer and the static parameter is directly used as $\mathbf{z}_p$. The static encoder outputs a d_{zp}-dimensional vector $\mathbf{z}_p \in \mathbb{R}^{d_{zp}}$.

Latent evolution model g. The latent evolution model g takes as input the concatenation of $\mathbf{z}^k$ and $\mathbf{z}_p$ (concatenated along the feature dimension), and outputs the prediction $\hat{\mathbf{z}}^{k+1}$. We model it as an MLP with residual connection from input to output, as an equivalent of the forward Euler's method in latent space:

$$\hat{\mathbf{z}}^{k+1} = \text{MLP}_g(\mathbf{z})^k + \mathbf{z}^k \tag{15}$$

In this work, we use the same MLP_g architecture throughout all sections, where the MLP_g consists of 5 layers, each layer has the same number d_z of neurons as the dimension of $\mathbf{z}^k$. The first three layers has ELU activation, and the last two layers have linear activation. We use two layers of linear layer instead of one, to have an implicit rank-minimizing regularization [76], which we find performs better than 1 last linear layer.

Decoder h. Mirroring the encoder q, the decoder h takes as input the $\mathbf{z}^{k+m} \in \mathbb{R}^{d_z}, m = 0, 1, ...M$, through an MLP_h and a CNN with $F_h = F_q$ number of convolution-transpose blocks, and maps to the state U^{k+m} at input space. The MLP_h is a one layer MLP with linear activation. After it, the vector is reshaped into the shape of (batch-size, channel-size, *image-shape) for the F_h convolution-transpose blocks. Then it is followed by a single convolution-transpose layer with (kernel-size, stride, padding)=$(3, 1, 1)$ and linear activation. Each convolution-transpose block consists of one convolution-transpose layer with (kernel-size, stride, padding) = $(4, 2, 1)$, followed by group normalization and an ELU activation. The number of channels also follows a mirroring of the encoder q, where the nearer to the output, the smaller the channel size with exponentially decreasing size.

Limitations of current LE-PDE architecture. The use of MLPs in the encoder and decoder has its benefits and downside. The benefit is that due to its flatten operation and MLP that maps to a much smaller vector $\mathbf{z}$, it can significantly improve speed, as demonstrated in the experiments in the paper. The limitation is that it requires that the training and test datasets to have the same discretization, otherwise a different discretization will result in a different flattened dimension making the MLP in the encoder and decoder invalid. We note that despite this limitation, it already encompasses a vast majority of applications where the training and test datasets share the same discretization (but with novel initial condition, static parameter p, etc.). Experiments in this paper show that our method is able to generalize to novel equations in the same family (Sec. 4.1), novel initial conditions (Sec. 4.2 and 4.3) and novel Reynolds numbers in 3D (Appendix F). Furthermore, our general 4-component architecture of dynamic encoder, static encoder, latent evolution model and decoder is very general and can allow future work to transcend this limitation. Future work may go beyond the limitation of discretization, by incorporating ideas from *e.g.* neural operators [34, 36], where the latent vector encodes the solution *function* $\mathbf{u}(\mathbf{x}, t)$ instead of the discretized states U^k, and the latent evolution model then models the latent dynamics of neural *operators* instead of functions.

Similar to a majority of other deep-learning based models for surrogate modeling (*e.g.* [13, 14]), the conservation laws present in the PDE is *encouraged* through the loss w.r.t. the ground-truth, but not generally *enforced*. Building domain-specific architectures that enforces certain conservation laws is out-of-scope of this work, since we aim to introduce a more general method for accelerating simulating and inverse optimizing PDEs, applicable to a wide scope of temporal PDEs. It is an exciting open problem, to build more structures into the latent evolution that obeys certain conservation laws or symmetries, potentially incorporating techniques *e.g.* in [77, 78]. Certain conservation laws can also be enforced in the decoder, for example similar to the zero-divergence as in [57].

D Details for experiments in 1D family of nonlinear PDEs

Here we provide more details for the experiment for Sec. 4.1. The details of the dataset have already been given in Section 4.1 and more detailed information can be found in [7] that introduced the benchmark.

LE-PDE. For LE-PDE in this section, the convolution and convolution-transpose layers are 1D convolutions, since the domain is 1D. We use temporal bundling steps $S = 25$, similar to the MP-PDE, so it based on the past $S = 25$ steps to predict the next $S = 25$ steps. The input has shape of (batch-size, S, $C_{\text{in}} = 1$, n_x) , which[3] we flatten the S and C_{in} dimensions into a single dimension and feed the (batch-size, $S \times C_{\text{in}} = 25$, n_x) tensor to the encoder. For the convolution layers in encoder, we use starting channel size $C = 32$ and exponential increasing channels as detailed in Appendix C. We use $F_q = F_r = 4$ blocks of convolution (or convolution-transpose).

We perform search on hyperparameters of latent dimension $d_z \in \{64, 128\}$, loss function $\ell \in \{\text{MSE}, \text{RMSE}\}$, time horizon $M \in \{4, 5\}$, and number of layers for static encoder $F_r \in \{0, 1, 2\}$, and use the model with the best validation loss. We train for 50 epochs with Adam [61] optimizer with learning rate of 10^{-3} and cosine learning rate annealing [76] whose learning rate follows a cosine curve from 10^{-3} to 0.

Baselines. For baselines, we directly report the baselines of MP-PDE, FNO-RNN, FNO-PR and WENO5 as provided in [7]. Details for the baselines is summarized in Sec. 4.1 and more in [7].

More explanation for Table 1. The runtimes in Table 1 are for one full unrolling that predicts the future 200 steps starting at step 50, on a NVIDIA 2080 Ti RTX GPU. The "full" runtime includes the time for encoder, latent evolution, and decoding to all the intermediate time steps. The "evo" runtime only includes the runtime for the encoder and the latent evolution. The representation dimension, as explained in Sec. 4.1, is the number of feature dimensions to update at each time step. For baselines of MP-PDE, etc. it needs to update $n_x \times S \times 1$ dimensions, *i.e.* the consecutive $S = 25$ steps of the 1D space with n_x cells (where each cell have one feature). For example, for $n_x = 100$, the representation dimension is $n_x \times S \times 1 = 100 \times 25 \times 1 = 2500$. In contrast, our LE-PDE uses a 64 or 128-dimensional latent vector to represent the same state, and only need to update it for every latent evolution.

Visualization of LE-PDE rollout. In Fig. 4, we show example rollout of our LE-PDE in the **E2** scenario and comparing with ground-truth. We see that LE-PDE captures the shock formation (around $x = 14$) faithfully, across all three spatial discretizations.

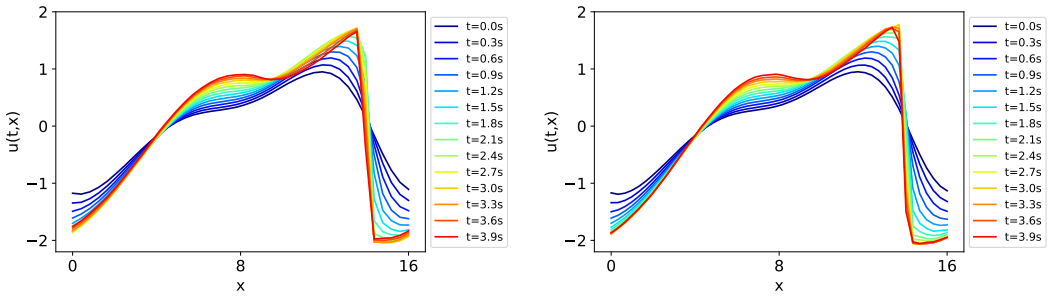
E Details for 2D Navier-Stokes flow

Here we detail the experiments we perform for Sec. 4.2. For the baselines, we use the results reported in [14]. For our LE-PDE, we follow the same architecture as detailed in Appendix C. Similar to other models (*e.g.* FNO-2d), we use temporal bundling of $S = 1$ (no bundling) and use the past 10 steps to predict one future step, and autoregressively rollout for $T - 10$ steps, then use the relative L2 loss over the all the predicted states as the evaluation metric. We perform search on hyperparameters of latent dimension $d_z \in \{128, 256\}$, loss function $\ell \in \{\text{MSE}, \text{RMSE}, \text{L2}\}$, time horizon $M \in \{4, T - 10\}$, number of epochs $\{200, 500\}$, and use the model with the best validation loss. The runtime in Table 2 is computed using an Nvidia Quadro RTX 8000 48GB GPU (since the FNO-3D exceeds the memory of the Nvidia 2080 Ti RTX 11GB GPU, to make a fair comparison, we use this larger-memory GPU for all models for runtime comparison).

F 3D Navier-Stokes flow

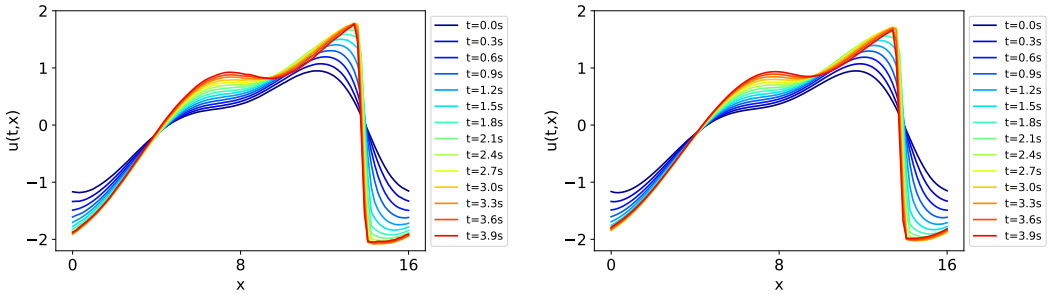
To explore how LE-PDE can scale to larger scale turbulent dynamics and its potential speed-up, we train LE-PDE in a 3D Navier-Stokes flow through the cylinder using a similar 3D dataset in [12], generated by PhiFlow [73] as the ground-truth solver. The PDE is given by:

[3] Here C_{in} is the number of input channels for $u(t, x)$. It is 1 since the $u(t, x)$ has only one feature.



(a) LE-PDE prediction with $n_x = 40$

(b) LE-PDE prediction with $n_x = 50$



(c) LE-PDE prediction with $n_x = 100$

(d) Ground-truth with $n_x = 100$

Figure 4: Example rollout of LE-PDE for 200 steps (0 to 4s), with **E2** scenario that tests the models ability to generalize to new equations within the same family, for (a) $n_x = 40$, (b) $n_x = 50$, (c) $n_x = 100$, compared with ground-truth of (d) $n_x = 100$. The LE-PDE models in the plot are using the ones reported in Table 1. We see that LE-PDE captures the shock formation (around $x = 14$) very accurately and faithfully, across all three spatial discretizations.

$$\partial_t u_x + \mathbf{u} \cdot \nabla u_x = -\frac{1}{\rho} \nabla p + \nu \nabla \cdot \nabla u_x, \tag{16}$$

$$\partial_t u_y + \mathbf{u} \cdot \nabla u_y = -\frac{1}{\rho} \nabla p + \nu \nabla \cdot \nabla u_y, \tag{17}$$

$$\partial_t u_z + \mathbf{u} \cdot \nabla u_z = -\frac{1}{\rho} \nabla p + \nu \nabla \cdot \nabla u_z, \tag{18}$$

$$\text{subject to } \nabla \cdot \mathbf{u} = 0. \tag{19}$$

We discretize the space into a 3D grid of $256 \times 128 \times 128$, resulting in 4.19 million cells per time step. We generate 5 trajectories of length 500 with Reynolds number $\{55.5, 56.8, 58.0, 58.3, 58.6\}$ for training/validation set and test the model's performance on 2 additional trajectories with $\{57.4, 58.0\}$. All the trajectories have different initial conditions. We sub-sample the time every other step, so the time interval between consecutive time step for training is 2s. For LE-PDE, we follow the architecture in Appendix C, with $F_q = F_h = 5$ convolution (convolution-transpose) blocks in the encoder (decoder), latent dimension $d_z = 128$, and starting channel dimension of $C = 32$. We use time horizon $M = 4$ in the learning objective (Eq. 5), with $(\alpha_1, \alpha_2, \alpha_3, \alpha_4) = (1, 0.1, 0, 0.1)$ (we set the third step $\alpha_3 = 0$ due to the limitation in GPU memory). The Reynolds number $p = Re$ is copied 4 times and directly serve as the static latent parameter (number of layers F_r for static encoder MLP r is 0). This static encoder allows LE-PDE to generalize to novel Reynolds numbers. We use ℓ =MSE. We randomly split 9:1 in the training/validation dataset of 5 trajectories, train for 30 epochs, save the model after each epoch, and use the model with the best validation loss for testing.

Prediction quality. In Fig. S1, we show the prediction of LE-PDE on the first test trajectory with a novel Reynolds number ($Re = 57.4$) and novel initial conditions. We see that LE-PDE captures the high-level and low-level turbulent dynamics in a qualitatively reasonable way, both at the tail and also

in the inner bulk. This shows the scalability of our LE-PDE to learn large-scale PDEs with intensive dynamics in a reasonably faithful way.

Speed comparison. We compare the runtime of our LE-PDE, an ablation LE-PDE-~~latent~~ and the ground-truth solver PhiFlow, to predict the state at $t = 40$. The result is shown in Table 5. For the ablation LE-PDE-~~latent~~, its latent evolution model and the MLPs in the encoder and decoder are ablated, and it directly uses the other parts of encoder and decoder to predict the next step (essentially a 12-layer CNN). We see that our LE-PDE achieves a $70.80/0.084 \simeq 840\times$ speed up compared to the ground-truth solver on the same GPU. We see that w.r.t. LE-PDE-~~latent~~ (a CNN) that is significantly faster than solver, our LE-PDE is still $1.03/0.084 = 12.3$ times faster. This shows that our LE-PDE can significantly accelerate the simulation of large-scale PDEs.

Comparison of number of parameters. We see that our LE-PDE uses much less number of parameters to evolve autoregressively than FNO. The most parameters of LE-PDE are mainly in the encoder and decoder, which is only applied once at the beginning and end of the evolution. Thus, LE-PDE achieves a much smaller runtime than FNO to evolve to t=40.

G Details for inverse optimization of boundary conditions

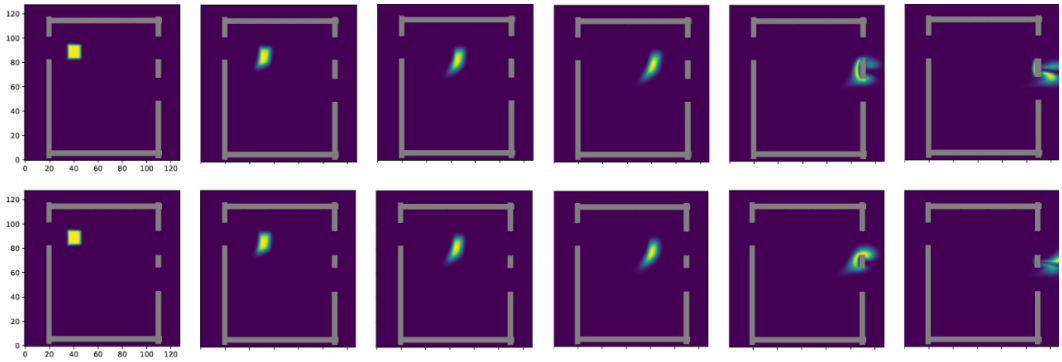


Figure S4: Trajectories generated by ground-truth solver with initial boundary parameter (upper) and optimized boundary parameter (lower).

Objective function. To define the objective function, we create masks (o_1, o_2) that correspond to respective outlets of given a boundary. The masks are defined to be ones on the outlets' voids (see also Fig. S5). With the masks, we define the objective function in Sec. 3.3 that can measure the amount of smoke passing through the outlets:

$$L_d[p] = \sum_{i=1}^{2} \mathrm{MSE}(t_i, \frac{\sum_{m=k_s}^{k_e} \langle o_i, \hat{U}^m(p) \rangle}{K}).$$

Here, $(t_1, t_2) = (0.3, 0.7)$, $K = \sum_{j=1}^{2} \sum_{m=k_s}^{k_e} \langle o_j, \hat{U}^m(p) \rangle$ and $\langle x, y \rangle = x^{\mathrm{T}} y$. We set $k_s = 50$, *i.e.*, we use smoke at scenes after 50 time steps to calculate the amount of the smoke.

LE-PDE. The encoder q and decoder h have $F_q = F_h = 4$ blocks of convolution (or convolution-transpose) followed by MLP, as specified in Appendix C. The time step of input is set to be 1.

Table 5: Comparison of LE-PDE with baseline on runtime and representation dimension, in the 3D Navier-Stokes flow. The runtime is to predict the state at $t = 40$.

	Runtime (s)	Representation dimension	Error at $t = 40$	# Paramters	# Parameters for evolution model	Training time (min) per epoch	Memory usage (MiB)
PhiFlow (ground-truth solver) on CPU	1802	16.76×10^6	-	-	-	-	-
PhiFlow (ground-truth solver) on GPU	70.80	16.76×10^6	-	-	-	-	-
FNO (with 2-step loss)	7.00	16.76×10^6	**0.1695**	3,281,864	3,281,864	102	25,147
FNO (with 1-step loss)	7.00	16.76×10^6	0.3215	3,281,864	3,281,864	58	24,891
LE-PDE-~~latent~~	1.03	16.76×10^6	0.1870	71,396,976	71,396,976	69	21,361
LE-PDE (ours)	**0.084**	**128**	0.1947	65,003,120	83,072	65	25,595

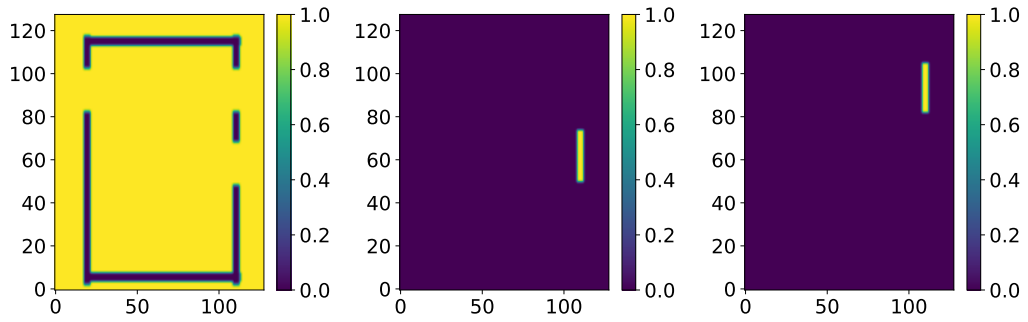
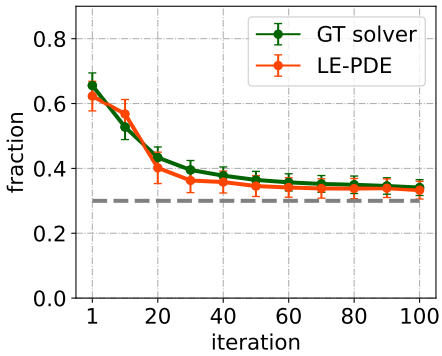


Figure S5: Figures of outlet masks for given a boundary mask. The left mask is a boundary mask, the middle mask o_1 corresponds to the lower outlet and the right o_2 the upper outlet.

The output of q is a 128-dimensional vector $\mathbf{z}^k$. The latent evolution model g takes as input the concatenation of $\mathbf{z}^k$ and 16-dimensional latent boundary representation $\mathbf{z}_p$ along the feature dimension, and outputs the prediction of $\hat{\mathbf{z}}^{k+1}$. Here, $\mathbf{z}_p$ is transformed by r with the same layers as q, taking as input an boundary mask, where the boundary mask is a interpolated one specified in Appendix B. The architecture of the latent evolution model g is the same as stated in Appendix C, with latent dimension $d_z = 128$.

Parameters for inverse design. We randomly choose 50 configurations for initial parameters. The sampling space is defined by the product of sets of inlet locations $\{79, 80, 81\}$, lower outlet locations $\{44, 45, 46, 47, 48, 49, 50\}$ and smoke position $\{0, 1\} \times \{-1, 0, 1\}$. We note that, even though we use the integers for the initial parameters, we can also use continuous values as initial parameters as long as the values are within the ranges of the integers. For one initial parameter, the number of the iterations of the inverse optimization is 100. During the iteration for each sampled parameter, we also perform linear annealing for β of continuous boundary mask starting from 0.1 to 0.05. We also perform an ablation experiment with fixed $\beta = 0.05$ across the iteration. Fig. S6 shows the result. We see that without annealing, the GT-solver (ground-truth solver) computed Error (0.041) is larger than with annealing (0.035), and the gap estimated by the model and the GT-solver is much larger. This shows the benefit of using boundary annealing.



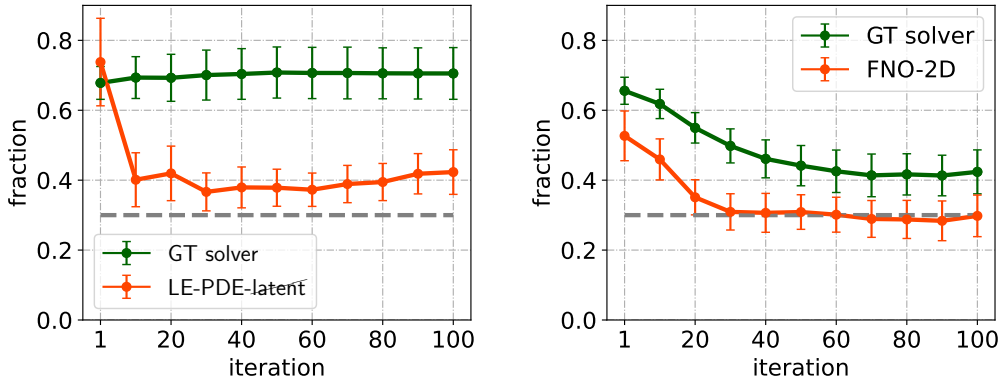
(a) Transition of fraction estimated by LE-PDE with fixed β. The difference (0.009) from fraction estimated by GT-solver is larger than that of LE-PDE with annealing (0.001) in Table 3.

LE-PDE (ours)	GT-solver Error (Model estimated Error)
constant β	0.041 (0.032)
linear annealing β	**0.035** (0.036)

(b) Fractions estimated by ablated version of the inverse optimizer. Continuous boundary parameter β in the ablated version is fixed across the iteration.

Figure S6: Ablation study of annealer in the inverse design for continuous boundary parameter β.

Model architecture of baselines. We use the same notation used in Appendix C. LE-PDE-~~latent~~ uses the dynamic encoder q subsequently followed by the decoder h. Both q and h have the same number of layers $F_q = F_h = 4$. The output of h is used as the input of the next time step. For the



(a) LE-PDE-~~latent~~ (b) FNO-2D

Figure S7: Fraction of smoke passing through the lower outlet computed by GT solver and estimated by LE-PDE-~~latent~~ and FNO-2D in Sec. 4.3. The dashed line denotes the objective of 0.3 fraction of smoke passing through the lower outlet.

FNO-2D model, we use the same architecture proposed in [14] with modes $= 12$ and width $= 20$. Fig. S7a and S7b are transition of fractions estimated by the ground-truth solver and the models with the boundary parameter under the inverse design. Compared with the one by our LE-PDE in Fig. 3e, we see that LE-PDE has much better GT-solver estimated fraction, and less gap between the fraction estimated by the GT-solver and the model.

H More ablation experiments with varying latent dimension

In this section, we provide complementary information to Sec. 4.4. Specifically, we provide tables and figures to study how the latent dimension d_z influences the rollout error and runtime. Fig. 6 visualizes the results. Table 6 shows the results in the 1D **E2** $(n_t, n_x) = (250, 50)$ scenario that evaluate how LE-PDE is able to generalize to novel PDEs within the same family. And Table 7 shows the results in the 2D most difficult $(\nu = 10^{-5}, N = 1000)$ scenario.

1D dataset. From Table 6 and Fig. 6a, we see that when latent dimension d_z is between 16 and 128, the accumulated MSE is near the optimal of $1 \sim 1.1$. It reaches minimum at $d_z = 64$. With larger latent dimensions, *e.g.* 256 or 512, the error slightly increases, likely due to the overfitting. With smaller latent dimension (< 8), the accumulated error grows significantly. This shows that the intrinsic dimension of this 1D problem with temporal bundling of $S = 25$ steps, is somewhere between 4 and 8. Below this intrinsic dimension, the model severely underfits, resulting in huge rollout error.

From the "runtime full" and "runtime evo" columns of Table 6 and also in Fig. 6b, we see that as the latent dimension d_z decreases down from 512, the "runtime evo" has a slight decreasing trend down to 256, and then remains relatively flat. The "runtime full" also remains relatively flat. We don't see a significant decrease in runtime with decreasing d_z, likely due to that the runtime does not differ much in GPU with very small matrix multiplications.

2D dataset. From Table 7 and Fig. 6c, we see that similar to the 1D case, the error has a minimum in intermediate values of d_z. Specifically, as the latent dimension d_z decreases from 512 to 4, the error first goes down and reaching a minimum of 0.1861 at $d_z = 128$. Then it slightly increase with decreasing d_z until $d_z = 16$. When $d_z < 16$, the error goes up significantly. This shows that large latent dimension may results in overfitting, and the intrinsic dimension for this problem is somewhere between 8 and 16, below which the error will significantly go up. As the latent dimension decreases, the runtime have a very small amount of decreasing (from 512 to 256) but mostly remain at the same level. This relatively flat behavior is also likely due to that the runtime does not differ much in GPU with very small matrix multiplications.

Table 6: Performance evaluation for LE-PDE with different latent dimension on 1D dataset (E2-50 scenario). The accumulated error $= \frac{1}{n_x} \sum_{t,x}$ MSE, summing over the predicted steps of 50-250, the same as in Table 1. The runtime is measured by rolling out with the same 200 steps, measured on a NVIDIA 2080 Ti RTX GPU, same as in Table 1. The default is with $d_z = 128$.)

LE-PDE setting	**cumulative error**	**runtime (full) (ms)**	**runtime (evolution) (ms)**	**# parameters**	**# parameters for latent evolution model**
$d_z = 512$	2.778	16.3 ± 2.6	6.7 ± 1.0	4043648	1314816
$d_z = 256$	2.186	15.0 ± 0.8	6.1 ± 0.3	2271360	329728
$d_z = 128$	1.127	14.9 ± 1.1	6.0 ± 0.4	1630976	82944
$d_z = 64$	0.994	14.4 ± 1.0	5.7 ± 0.3	1372224	20992
$d_z = 32$	1.048	14.5 ± 0.8	5.8 ± 0.4	1258208	5376
$d_z = 16$	1.041	14.1 ± 0.9	5.8 ± 0.4	1205040	1408
$d_z = 8$	21.03	14.0 ± 0.7	5.6 ± 0.2	1179416	384
$d_z = 4$	205.09	13.9 ± 0.5	5.7 ± 0.3	1166844	112

Table 7: Performance evaluation for LE-PDE with different latent dimension on 2D dataset ($\nu = 10^{-5}$ scenario. The Error is the relative L2 norm measured over 10 rollout steps, the same as in Table 2. The runtime is measured by rolling out with the same 10 steps, measured on a Nvidia Quadro RTX 8000 48GB GPU (same as in Table 2), and average over 100 runs (the number after $\pm$ is the std. of the 100 runs). The default is with $d_z = 128$.)

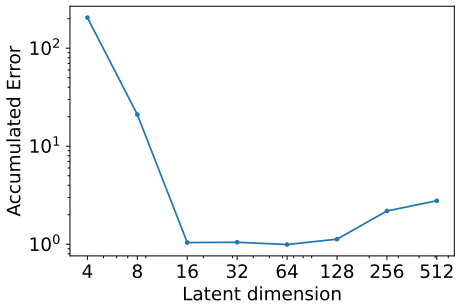
LE-PDE setting	**cumulative error**	**runtime (full) (ms)**	**runtime (evolution) (ms)**	**# parameters**	**# parameters for latent evolution model**
$d_z = 512$	0.1930	16.2 ± 1.1	6.8 ± 0.7	6467184	1313280
$d_z = 256$	0.1861	14.8 ± 1.1	5.8 ± 0.4	3384944	328960
$d_z = 128$	0.2064	14.8 ± 0.5	5.9 ± 0.4	2089584	82560
$d_z = 64$	0.2252	14.7 ± 0.7	6.0 ± 0.7	1503344	20800
$d_z = 32$	0.2315	15.0 ± 2.1	5.9 ± 0.5	1225584	5280
$d_z = 16$	0.2236	14.2 ± 1.3	5.8 ± 0.6	1090544	1360
$d_z = 8$	0.3539	14.3 ± 0.6	5.7 ± 0.3	1023984	360
$d_z = 4$	0.6353	14.2 ± 0.5	5.7 ± 0.2	990944	100

More details in the ablation study experiments in Sec. 4.4. For the ablation "Pretrain with L_{recons}", we pretrain the encoder and decoder with L_{recons} for certain number of epochs, then freeze the encoder and decoder and train the latent evolution model and static encoder with $L_{\text{consistency}}$. Here the $L_{\text{multi-step}}$ is not valid since the encoder and decoder are already trained and frozen. For both 1D and 2D, we search hyperparameters of pretraining with $\{25, 50, 100\}$, and choose the model with the best validation performance.

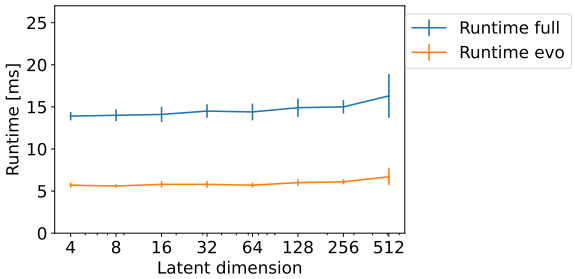
I Broader social impact

Here we discuss the broader social impact of our work, including its potential positive and negative aspects, as recommended by the checklist. On the positive side, our work have huge potential implication in science and engineering, since many important problems in these domains are expressed as temporal PDEs, as discussed in the Introduction (Sec. 1). Although this work focus on evaluating our model in standard benchmarks, the experiments in Appendix F also show the scalability of our method to problems with millions of cells per time steps under turbulent dynamics. Our LE-PDE can be applied to accelerate the simulation and inverse optimization of the PDEs in science and engineering, *e.g.* weather forecasting, laser-plasma interaction, airplane design, etc., and may significantly accelerate such tasks.

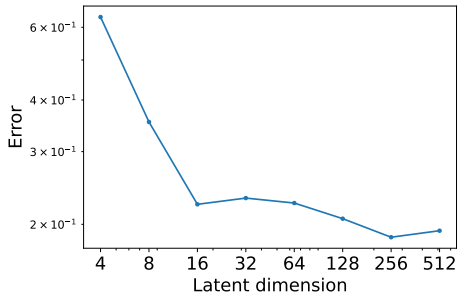
We see no obvious negative social impact of our work. As long as it is applied to the science and engineering that is largely beneficial to society, our work will have beneficial effect.



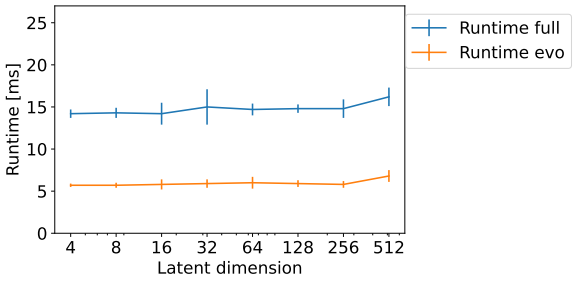
(a) Accumulated Error vs. latent dimension in 1D scenario. The Error is the relative L2 norm measured over 10 rollout steps, the same as in Table 2. The runtime is measured by rolling out with the same 10 steps, measured on a Nvidia Quadro RTX 8000 48GB GPU (same as in Table 2), and average over 100 runs (the number after $\pm$ is the std. of the 100 runs). The default is with $d_z = 128$.



(b) Runtime vs. latent dimension in 1D scenario



(c) Error vs. latent dimension in 2D scenario.



(d) Runtime vs. latent dimension in 2D scenario

Figure 6: Error vs. latent dimension d_z for (a) 1D and (c) 2D scenario, and runtime vs. latent dimension d_z for (b) 1D and (d) 2D scenario. We see that in 1D, the Error stays near optimum with latent size in [16, 128], and goes up outside the range. The runtime evo have a slight decreasing trend from latent dimension at 512 to 256, and stays relatively flat. For 2D, the Error decreases with increasing latent dimension, reaching an optimum at $d_z = 256$, and then slightly increases. Its runtime full have a slight decrease from latent dimension of 512 down to 256, and otherwise stays relatively flat.

J Pareto efficiency of FNO vs. LE-PDE

The following Table S8 shows the comparison of performance of FNO with varying hyperparameters. The hyperparameter search is performed on a 1D representative dataset E2-50. We evaluate the models (with varying hyperparameters) using the metric of the cumulative error and runtime. The most important hyperparameters for FNO are the "modes", which denotes the number of Fourier frequency modes, and "width", which denotes the channel size for the convolution layer in the FNO.

We also perform hyperparameter search on a 2D representative dataset with $\nu = 10^{-5}$. Table S9 shows the comparison of performance of FNO with varying hyperparameters. Hyperparameters to be varied and metrics for the evaluation are same as that of Table S8.

We can compare the above two tables with Table 6 and 7. We also create plots Figure S8 and S9 that compare the trade-off between several metrics shown in the tables for LE-PDE and FNO. Note that we provide both the total number of parameters (second last column) and number of parameters for latent evolution model (last column). The latter is also a good indicator since during long-term evolution, the latent evolution model is autoregressively applied while the encoder and decoder are only applied once. So the latent evolution model is the deciding component of the long-term evolution accuracy and runtime.

Table S8: Performance evaluation with FNO hyperparameter search on 1D dataset (E2-50 scenario.)

FNO setting	cumulative error	runtime (full) (ms)	# parameters
modes=16, width=64 (default setting)	2.379	21.2± 6.9	292249
modes=16, width=128	3.107	21.7± 4.3	1138201
modes=16, width=32	2.695	22.1± 7.4	78169
modes=16, width=16	2.755	21.0± 5.7	23353
modes=16, width=8	4.992	17.9± 1.2	9001
modes=20, width=128	2.804	20.9± 1.1	1400345
modes=20, width=64	2.626	19.3± 0.9	357785
modes=12, width=64	2.899	19.6± 2.2	226713
modes=8, width=64	2.240	19.7± 1.3	161177
modes=4, width=64	2.326	19.2± 0.9	95641
modes=8, width=32	2.366	18.2± 1.0	45401
modes=8, width=16	2.505	18.1± 1.2	15161
modes=8, width=8	5.817	18.4± 1.2	6953

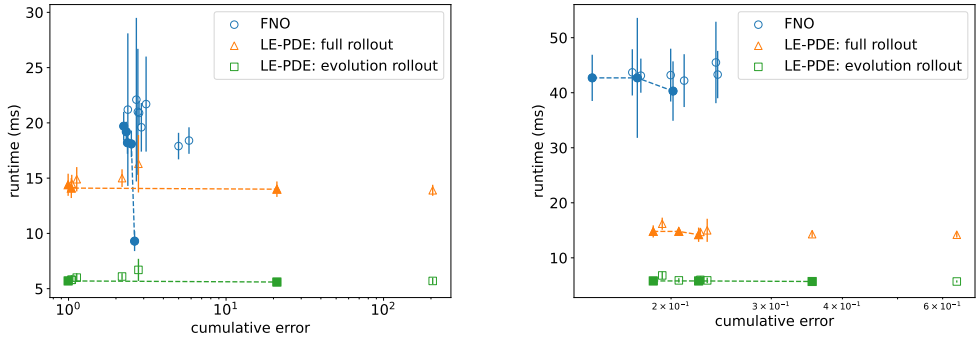
Table S9: Performance evaluation with FNO hyperparameter search on 2D dataset ($\nu = 10^{-5}$ scenario.)

FNO setting	L2 error	runtime (full)(ms)	# parameters
modes=12, width=20 (default setting)	0.1745	42.7 ± 10.9	465717
modes=12, width=40	0.1454	42.7 ± 4.2	1855977
modes=12, width=10	0.2016	40.3 ± 5.4	117387
modes=12, width=5	0.2398	45.5 ± 7.4	29922
modes=16, width=20	0.1710	43.7 ± 4.2	824117
modes=8, width=20	0.1770	43.1 ± 3.1	209717
modes=4, width=20	0.1997	43.2 ± 4.8	56117
modes=8, width=10	0.2109	42.2 ± 4.8	53387
modes=8, width=5	0.2415	43.3 ± 4.3	13922

From the comparison, we see that:

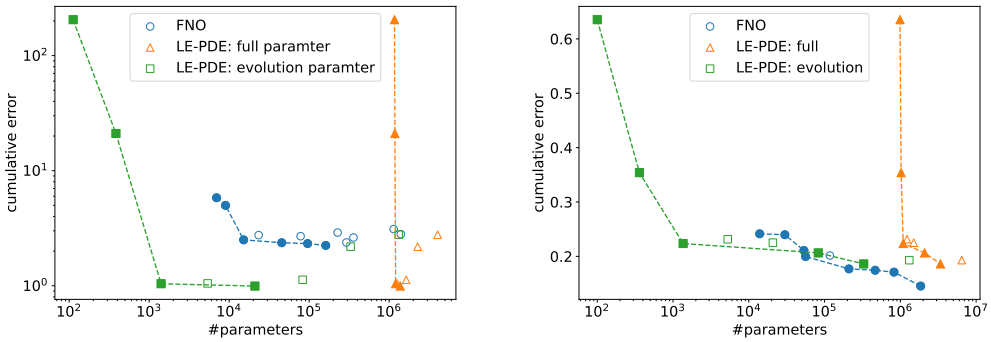
- For 1D dataset, LE-PDE Pareto-dominates FNO in error vs. runtime plot (Fig. S8(a)). FNO's best cumulative error is 2.240, and runtime is above 17.9ms, over the full hyperparameters combinations (number of parameter varying from 6953 to 1.4M). In comparison, our LE-PDE achieves much better error and runtime over a wide parameter range: for d_z from 16 to 64, LE-PDE's cumulative error $\leq$ 1.05, runtime $\leq$ 14.5ms, latent runtime $\leq$ 5.8ms, (which uses 1408 to 82944 number of parameters for latent evolution model, and 1.2-1.4M total parameters). In terms of cumulative error vs. #parameter plot (Fig. S9(a)), the LE-PDE with evolution model typically has less parameters than FNO, which in turn also have less parameters than LE-PDE with full model. This makes sense, as the latent evolution requires much less parameters. Adding the encoder and decoder, LE-PDE may have more #parameters. But still it is the evolution parameter that is the most important for long-term evolution.

- For 2D dataset, FNO's cumulative error is slightly better than LE-PDE, but its runtime is significantly larger (Fig. S8(b)). Concretely, the best FNO achieves an error of 0.1454 while the best LE-PDE's error is 0.1861. FNO's runtime is above 40ms, while LE-PDE's runtime is generally below 15ms and latent evolution runtime is below 6ms. LE-PDE uses larger total number of parameters but much less number of parameters for latent evolution model. Also, similar to 1D, in terms of error vs. #parameter plot (Fig. S9(b)), the LE-PDE with evolution model typically has much less parameters than FNO, which in turn also typically have less parameters than LE-PDE with full model.



(a) 1D dataset. (b) 2D dataset.

Figure S8: Comparison of trade-off between cumulative error and runtime of LE-PDE and FNO for 1D and 2D dataset. Dotted line connected to filled marker is Pareto frontier for respective model.



(a) 1D dataset. (b) 2D dataset.

Figure S9: Comparison of trade-off between number of parameters and cumulative error of LE-PDE and FNO for 1D and 2D dataset. Dotted line connected to filled marker is Pareto frontier for respective model.

Which family of PDEs can our LE-PDE apply:. we can think of a PDE as a ground-truth model that evolves the *state* of a physical system. Typically, the states show more global, dominant features, and can be described by a state vector with much smaller dimension than the original discretization. Our LE-PDE exploit this *compressivity of state* to evolve the system in latent space and achieve speedup, and as long as the PDE does not significantly increase the spatial complexity of the state as it evolves (e.g. developing finer and finer spatial details as in 2-stream instability in of plasma [75]), our method can apply. Most of the PDEs satisfy the above requirements that the state are compressible and does not significantly increase its complexity, so our LE-PDE can be apply to most PDEs. Since any compression of state can incur a possible increase of error (possibly large or small, as the Pareto frontier of "error vs. runtime" and "error vs. #parameter" in Fig. S8 and S9 show for our LE-PDE and FNO), the more important/relevant question is then "what is the tradeoff of error vs. runtime we want for the given PDE", since we can design the encoder of LE-PDE with varying amount of compression. For example, we can design an encoder with minimal compression, so runtime reduction is low but can guarantee to retain low error, or with a much more aggressive compression (like in our 2D and 3D experiments), but can still achieve minimal increase of error. The amount of compression is a hyperparameter which can be obtained via validation set. Theoretically studying the best amount of compression that achieves a good tradeoff will be left for an exciting future work.

K Comparison of LE-PDE with LFM

To compare our LE-PDE with the Latent Field Model method (LFM) proposed in [18], we perform additional experiments in the representative 1D and 2D datasets in Section 4.4. We perform the ablation study where we (a) remove MLP in our model, (b) use LFM objective but maintain MLP, and (c) full LFM: remove MLP, use LFM objective, while all other aspects of training is kept the same. We use PyTorch's jvp function in autograd to compute the Jacobian-vector product and carefully make sure that our implementation is correct. Table S10 is the comparison table.

Table S10: Performance comparison of LE-PDE with LFM, for 1D dataset E2-50 scenario.

LE-PDE setting	**cumulative error**	**runtime (full) (ms)**	**runtime (evolution) (ms)**	**# parameters**	**# parameters for latent evolution model**
LE-PDE (ours)	1.127	14.9 ± 1.1	6.0 ± 0.4	1630976	82944
(a) without MLP	7.930	17.2 ± 6.0	8.3 ± 0.4	2730368	1580544
(b) with LFM objective	58.85	15.7 ± 1.5	6.5 ± 0.6	1630976	82944
(c) full LFM: without MLP, with LFM objective	26.12	15.7 ± 1.3	8.4 ± 0.7	2730368	1580544

Table S11 shows the comparison result of LE-PDE with LFM obtained by performing additional experiments on the representative 2D dataset in Section 4.4.

Table S11: Performance comparison of LE-PDE with LFM, for 2D dataset $\nu = 1e\text{-}5$ scenario.

LE-PDE setting	**cumulative error**	**runtime (full) (ms)**	**runtime (evolution) (ms)**	**# parameters**	**# parameters for evolution model**
LE-PDE (ours)	0.1861	14.8 ± 1.1	5.8 ± 0.4	3384944	328960
(a) without MLP	0.2120	16.6 ± 2.2	9.2 ± 0.8	2126960	1181184
(b) with LFM objective	0.4530	15.8 ± 2.3	6.2 ± 0.6	3384944	328960
(c) full LFM: without MLP, with LFM objective	0.6315	16.2 ± 1.9	9.1 ± 0.4	2126960	1181184

From the above tables, we see that without MLP, it actually results in worse performance (ablation (a)), and with LFM objective, the error is larger, likely due to that the dataset are quite chaotic and LFM may not adapt to the large time range in these datasets.

L Influence of varying noise amplitude

Here, we perform additional experiments on how the noise affects the performance, on the representative 1D used in Section 4.4 "Ablation Study". Table S12 shows the results. Specifically, we add random fixed Gaussian noise to the training, validation and test sets of the dataset, with varying amplitude. The noise is independently added to each feature of the dataset. It is also "fixed" in the sense that once added to the dataset, the noise is freezed and not re-sampled. This mimics the real world setting where random observation noise can corrupt the observation and we never have the ground-truth data to train and evaluate from.

We also perform experiments similar to Section L on a 2D representative dataset used in Section 4.4 "Ablation Study". Table S13 shows the results.

Note that the value range of both datasets are within $[-2, 2]$. From Table S12, we see that LE-PDE's cumulative error stays excellent (≤ 1.456) with noise amplitude $\leq 10^{-3}$, much smaller than state-of-the-art MP-PDE's error of 1.63 and FNO-PF's 2.27. Even with noise amplitude of 10^{-2}, the LE-PDE's error of 2.612 still remains reasonable.

From Table S13, we see that LE-PDE is quite resilient to noise, with error barely increases for noise amplitude up to 2×10^{-2}, and only shows minimal increase at noise level of 10^{-1}. As a context, U-Net's error is 0.1982 and TF-Net's error is 0.2268 (Table 2 in main text).

In summary, in the 1D and 2D datasets, we see that LE-PDE shows good robustness to Gaussian noise, where the performance is reasonable where the ratio of noise amplitude to the value range can

Table S12: Evaluation of cumulative error of LE-PDE on 1D dataset (E2-50 scenario) with varying noise amplitude. The amplitude is the standard deviation of the diagonal Gaussian and the value range of the state $u(t, x)$ is within $[-2, 2]$.

Noise amplitude	**cumulative error**
0 (default)	1.127
10^{-5}	1.253
10^{-4}	1.268
10^{-3}	1.456
10^{-2}	2.612
2×10^{-2}	4.102
5×10^{-2}	9.228

Table S13: Evaluation of cumulative error of LE-PDE on 2D dataset ($\nu = 10^{-5}$ scenario) with varying noise amplitude. The amplitude is the standard deviation of the diagonal Gaussian and the value range of the state $u(t, x)$ is within $[-2, 2]$.

Noise amplitude	**cumulative error**
0 (default)	0.1861
10^{-5}	0.1880
10^{-4}	0.1862
10^{-3}	0.1866
10^{-2}	0.1897
2×10^{-2}	0.1875
5×10^{-2}	0.1910
10^{-1}	0.2012

go up to 0.25% in 1D and 2.5% in 2D. The smaller robustness in the 1D Burgers' dataset may be due to that it is a 200-step rollout and the noise may make the model uncertain about the onset of shock formation.

M Ablation of LE-PDE using pretrained autoencoder or VAE

In addition, we perform two ablation experiments that explore performing data reduction first and then learn the evolution in latent space: (a) pretrain an autoencoder with states from all time steps, then freeze the autoencoder and train the latent evolution model. This mimics the method in [79]. (b) the encoder and decoder of LE-PDE is replaced with a VAE, first pre-trained with ELBO on all time steps, then freeze the encoder and decoder and train the latent evolution model. All other aspects of the model architecture and training remains the same. The result is shown in the Table S14 and Table S15 for the 1D and 2D datasets in Section 4.4 of "Ablation study".

Table S14: Ablation study of LE-PDE using pretrained autoencoder or VAE, for 1D dataset (E2-50 scenario.)

LE-PDE setting	**Cumulative error**
LE-PDE (ours)	1.127
(a) pretrain autoencoder	1.952
(b) pretrained VAE	1.980

From Table S14 and S15, we see that performing pre-training results in a much worse performance, since the data reduction only focuses on reconstruction, without consideration for *which* latent state is best used for evolving long-term into the future. On the other hand, our LE-PDE trains the components jointly with a novel objective that not only encourages better reconstruction, but also long-term evolution accuracy both in latent and input space. We also see that VAE as data-reduction performs worse than autoencoder, since the dynamics of the system is deterministic, and having a stochasticity from the VAE does not help.

Table S15: Ablation study of LE-PDE using pretrained autoencoder or VAE, for 2D dataset ($\nu = 10^{-5}$ scenario.)

LE-PDE setting	Cumulative error
LE-PDE (ours)	0.1861
(a) pretrain autoencoder	0.2105
(b) pretrained VAE	0.2329

References

[74] Y. Wu and K. He, "Group normalization," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 3–19.

[75] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *International Conference on Learning Representations*, 2016, [Online]. Available: https://arxiv.org/abs/1511.07289.

[76] L. Jing, J. Zbontar *et al.*., "Implicit rank-minimizing autoencoder," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 14736–14746, 2020.

[77] A. A. Kaptanoglu, K. D. Morgan, C. J. Hansen, and S. L. Brunton, "Physics-constrained, low dimensional models for magnetohydrodynamics: First-principles and data-driven approaches," in *Physical Review E*, vol. 104, no. 1, p. 015206, 2021.

[78] S. Yang, X. He, and B. Zhu, "Learning physical constraints with neural projections," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 5178–5189, 2020.

[79] I. Loshchilov and F. Hutter, "SGDR: Stochastic gradient descent with warm restarts," in *International Conference on Learning Representations (Poster)*, 2017, [Online]. Available: https://arxiv.org/abs/1608.03983.