Learning to accelerate simulation and inverse optimization of PDEs via latent global evolution

NeurIPS 2022

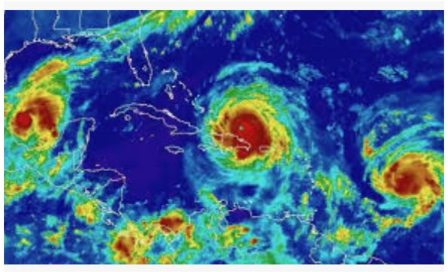
Tailin Wu[1], Takashi Maruyama[1,2], Jure Leskovec[1]

[1] Stanford University
[2] NEC

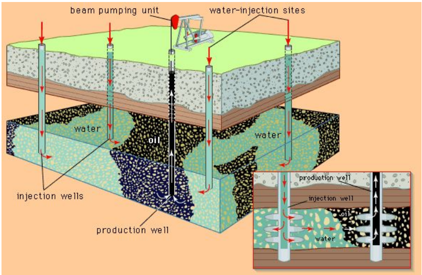
Outline

1. Introduction and preliminaries
2. Method
   - Forward simulation
   - Inverse optimization
3. Experiments

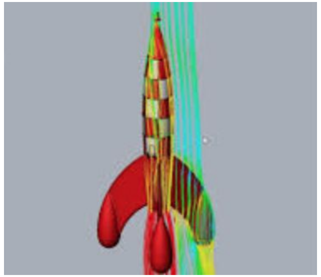
1.1 Partial differential equations (PDEs) are important in science and engineering



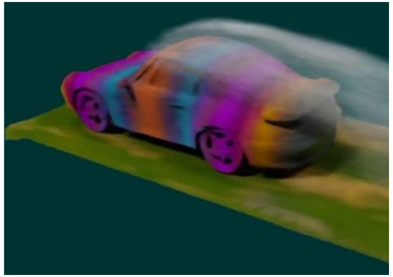
Weather forecasting



Subsurface fluid simulation for oil production



Aerodynamics for Rocket



Equipment manufacturing

Characteristics:

- Large-scale in size: state dimension of more than millions per time step
- Slow to simulate (requiring up to High Performance Computing, HPC)

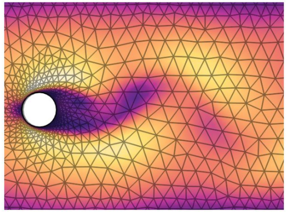
1.2 Classical solvers vs. deep learning-based surrogate models

Classical solvers:

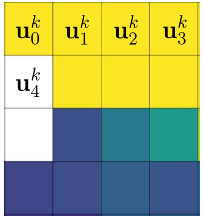
Based on Partial Differential Equations (PDEs)

$$\frac{\partial \mathbf{u}}{\partial t} = F(x, \mathbf{u}, \frac{\partial \mathbf{u}}{\partial \mathbf{x}}, \frac{\partial^2 \mathbf{u}}{\partial \mathbf{x}^2}, \ldots)$$

$\mathbf{u}:$ state
$\mathbf{x}:$ spatial coordinate
$t:$ time



mesh



grid

Discretize the PDE, then use finite difference, finite element, *etc*. to evolve the system

$\mathbf{u}(t, \mathbf{x}) \longrightarrow \mathbf{u}_i^k$

discrete time index k
discrete cell id i

Pros and challenges:

- **Pros:** (1) First principle-based and interpretable, (2) accurate, (3) have error guarantee.
- **Challenges: Slow**. Why:
  (a) To ensure numerical stability, typically have to use small time intervals
  (b) Sometimes have to use *implicit method* to ensure numerical stability, thus need to solve millions of implicit equations

1.2 Classical solvers vs. deep learning-based surrogate models

Deep learning-based surrogate models (e.g. [1-6]):

Pros:

- Directly learn from data, alleviating much engineering efforts.
- Offer speedup via larger spatial/temporal intervals and explicit forward

However, they typically evolve the system in the input space, which can still be **slow** and need huge computation (e.g. for millions cells, need to update each cell at each time step)

Prior reduce-order modeling methods [7-11] are limited in expressivity and scope

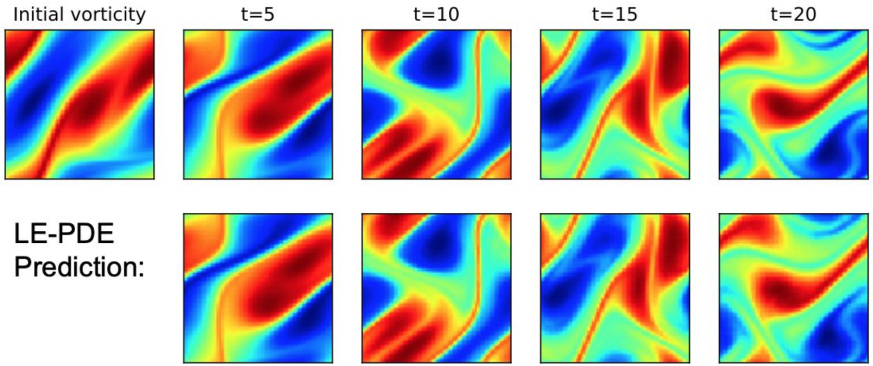
[1] Brandstetter, Johannes, Daniel Worrall, and Max Welling. "Message passing neural PDE solvers." ICLR (2022).
[2] Li, Zongyi, et al. "Fourier neural operator for parametric partial differential equations." ICLR (2020).
[3] L.Lu, et al. “Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. Nature Machine Intelligence 3 (3), 218–229 (2021).”
[4] Z.Li, et al., “Neural operator: Graph kernel network for partial differential equations,” arXiv preprint arXiv:2003.03485, 2020.
[5] Kochkov, Dmitrii, et al. "Machine learning–accelerated computational fluid dynamics." Proceedings of the National Academy of Sciences 118.21 (2021): e2101784118.
[6] 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,” NeurIPS 2020

[7] Treuille et al. 2006
[8] Kim et al. 2013
[9] Wiewel et al 2019
[10] Lee et al. 2020
[11] Vlachas et al. 2022

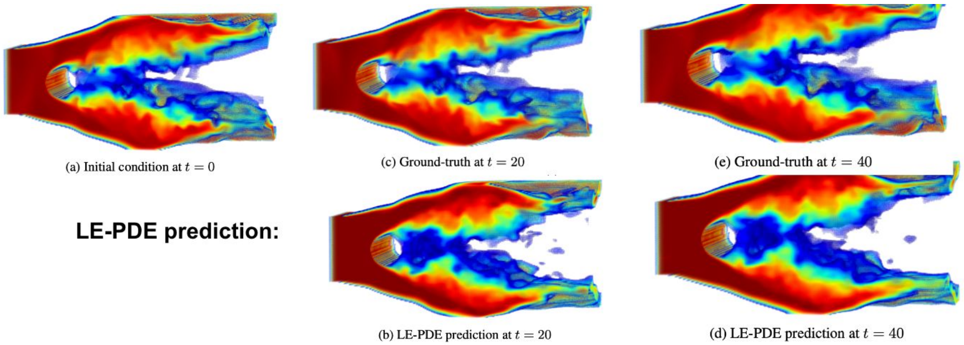
1.3 Present work: Latent Evolution of PDEs (LE-PDE)

We introduce a simple, fast and scalable method to **accelerate** the *forward simulation* and *inverse optimization* of PDEs, that achieves up to 15x speedup w.r.t. state-of-the-art deep learning-based models with competitive accuracy.

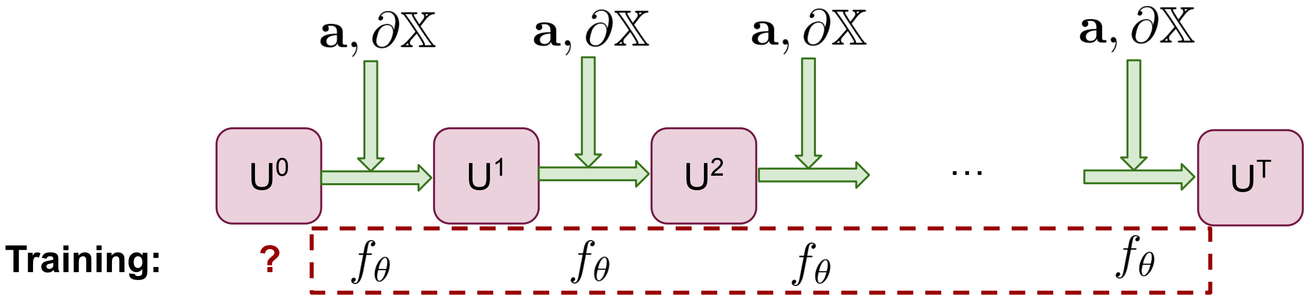
2D turbulent NS equation:



3D turbulent NS equation (4 millions cells):



2.1 Prior methods: forward simulation



Inference: Evolve the system in input space

U^t: discretized state of the system at time t

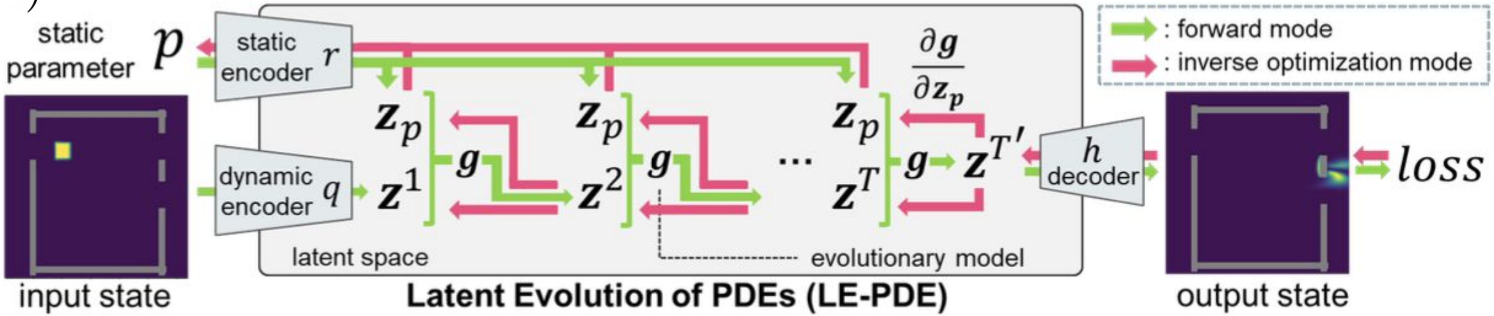
a: static parameters of the system that does not change with time
(e.g. parameters of PDE, spatially varying diffusion coefficient)

$\partial\mathbb{X}$: boundary condition of the system

f_θ: model to be learned

2.2 LE-PDE: architecture

$p := (\mathbf{a}, \partial\mathbb{X})$



Architecture:

q: $U^t \to \mathbf{z}^t$, dynamic encoder (CNN + flatten + MLP, can also be GNN + MLP for general mesh).

$\mathbf{z}^t$ is a "global" vector.

r: p $\to \mathbf{z}_p$, static encoder (CNN + flatten + MLP),

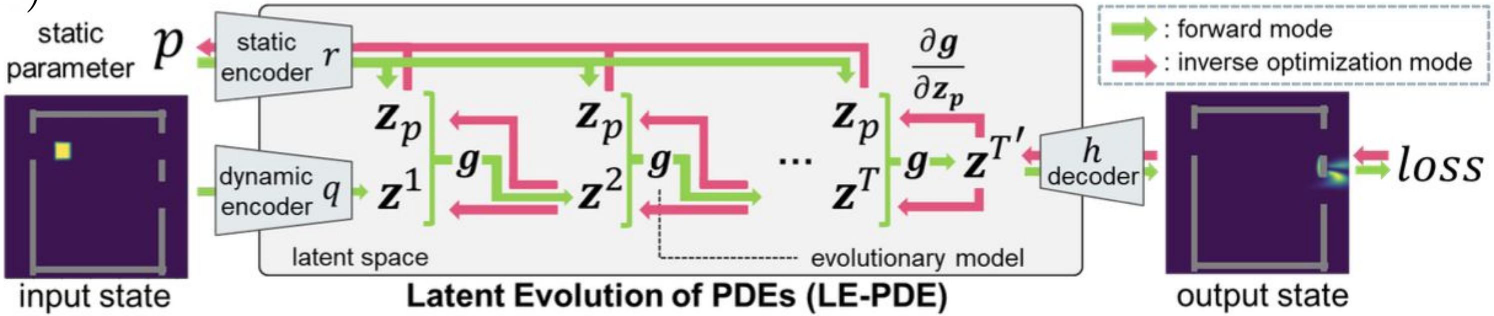
g: $(\mathbf{z}^t, \mathbf{z}_p) \to \mathbf{z}^{t+1}$ latent evolution model (MLP)

h: $\mathbf{z}^{t+k} \to U^{t+k}$, decoder (MLP + CNN with ConvTranspose)

Typically $\mathbf{z}^k$ has much smaller dimension than U^k, and the latent evolution model g has much less compute than evolving it in input space, thus achieving **speedup**.

2.3 LE-PDE: learning

$p := (\mathbf{a}, \partial\mathbb{X})$



Learning objective:

All 4 components are trained jointly from scratch

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

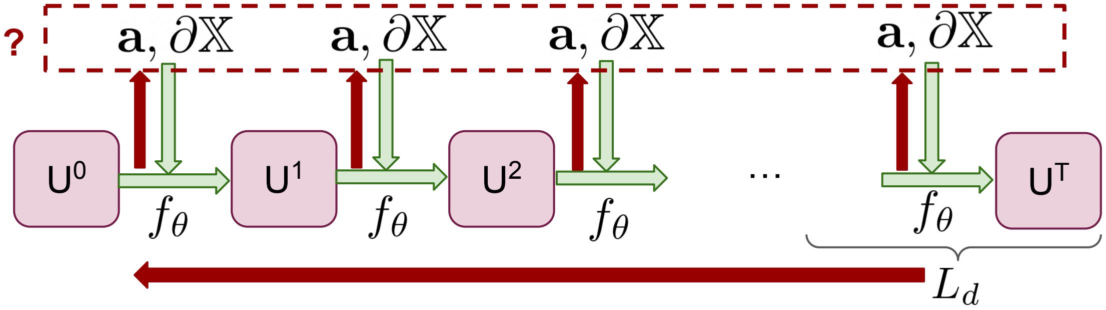
$$\text{where} \begin{cases} L^k_{\text{multi-step}} = \sum_{m=1}^{M} \alpha_m \ell(\hat{U}^{k+m}, U^{k+m}), \\ L^k_{\text{recons}} = \ell(h(q(U^k)), U^k) \\ L^k_{\text{consistency}} = \sum_{m=1}^{M} \frac{||g(\cdot, r(p))^{(m)} \circ q(U^k) - q(U^{k+m})||_2^2}{||q(U^{k+m})||_2^2} \end{cases}$$

(encourage long-term rollout in input space)

(reconstruction loss)

(long-term consistency in latent space)

2.4 Inverse optimization: prior methods



Optimize boundary $\partial\mathbb{X}$

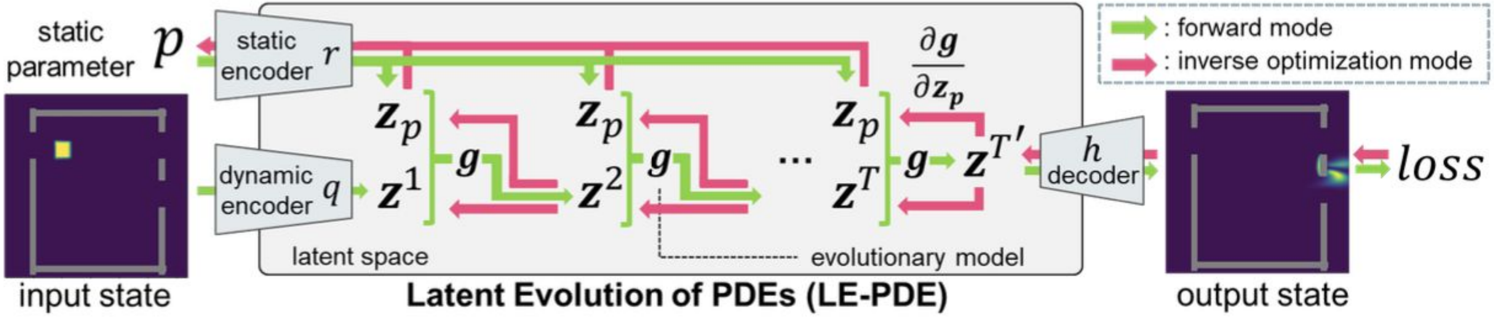
$$\partial\mathbb{X}^* = \mathrm{argmin}_{\partial\mathbb{X}} L_d[\mathbf{a}, \partial\mathbb{X}] = \mathrm{argmin}_{\partial\mathbb{X}} \sum_{m=k_s}^{k_e} \ell_d\left(\hat{U}^m(\mathbf{a}, \partial\mathbb{X})\right)$$

using gradient descent:

$$\frac{\partial L_d[\mathbf{a}, \partial\mathbb{X}]}{\partial(\partial\mathbb{X})}$$

[1] Allen, Kelsey R., et al. "Physical Design using Differentiable Learned Simulators." *arXiv preprint arXiv:2202.00728* (2022).
[2] Zhao, Qingqing, et al. "Learning to Solve PDE-constrained Inverse Problems with Graph Networks." arXiv preprint arXiv:2206.00711 (2022).

2.4 Inverse optimization: LE-PDE



Optimize boundary $\partial\mathbb{X}$

$$\partial\mathbb{X}^* = \text{argmin}_{\partial\mathbb{X}} L_d[\mathbf{a}, \partial\mathbb{X}] = \text{argmin}_{\partial\mathbb{X}} \sum_{m=k_s}^{k_e} \ell_d\left(\hat{U}^m(\mathbf{a}, \partial\mathbb{X})\right)$$

using gradient descent:

$$\frac{\partial L_d[\mathbf{a}, \partial\mathbb{X}]}{\partial(\partial\mathbb{X})}$$

Since LE-PDE performs the forward simulation in latent space, it can speed up inverse optimization by speeding up the inner loop's forward simulation

3. Experiments

We aim to evaluate the following 4 aspects:

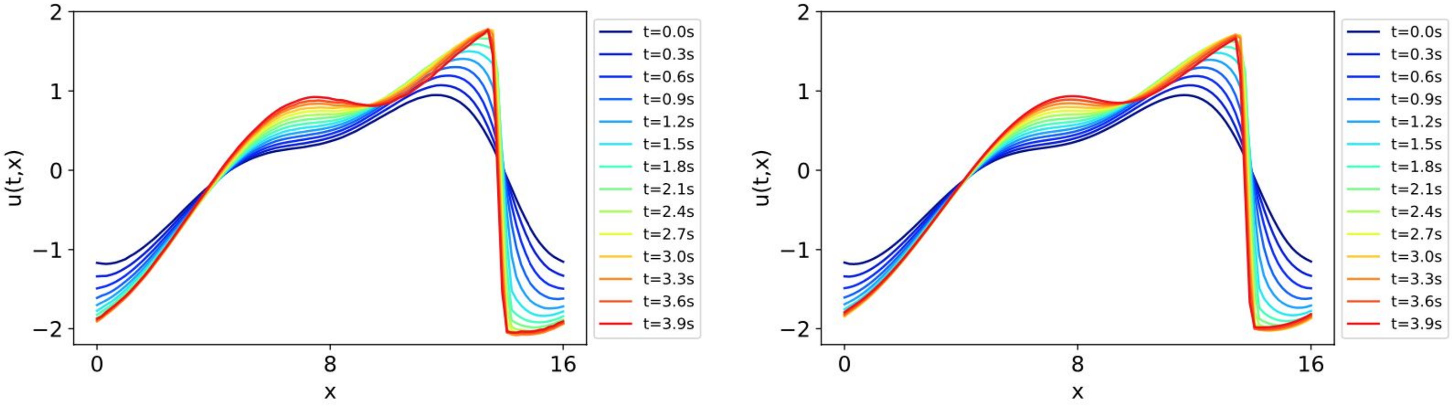
(1) **Accuracy:** does LE-PDE able to learn accurately the long-term evolution of challenging systems, and compare competitively with state-of-the-art methods?

(2) **Speed and scalability:** How much can LE-PDE reduce representation dimension and improving speed, especially with larger systems?

(3) **Inverse optimization:** Can LE-PDE improve and speed up inverse optimization?

3.1. 1D family of nonlinear PDEs

LE-PDE prediction (E2 scenario), starting at k=50 and predict next 200 steps



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

Challenge:
(1) long-term rollout (200 steps)
(2) Model the shock formation (near x=14)
(3) Also test the model's generalization to novel PDE parameters

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

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

3.1. 1D family of nonlinear PDEs

We compare state of-the-art models of Fourier Neural Operator (FNO), Message passing Neural PDE solver (MP-PDE), and a classical solver of WEBO5. FNO-RNN is from their original paper, FNO-PF is augmented with push-forward trick and temporal bundling:

		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**

Our LE-PDE achieves competitive accuracy, and speed up by up to 15x (compared to MP-PDE), with up to 2500/64=39 fold compression

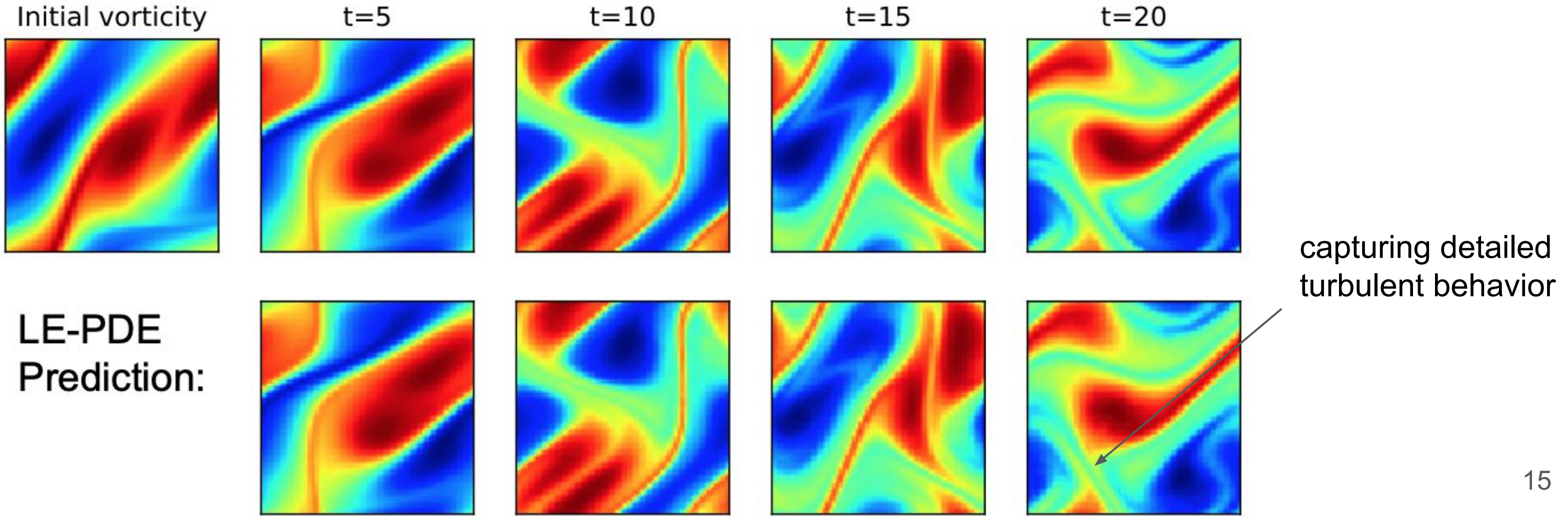
3.2. 2D Navier-Stokes Equation

PDE:

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

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

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



3.2. 2D Navier-Stokes Equation

Results:

N: number of training examples; T: total number of time steps; ν: viscosity for the N-S fluid

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 $[ms]$		Turbulent		
				$\nu = 10^{-3}$ $T = 50$ $N = 1000$	$\nu = 10^{-4}$ $T = 30$ $N = 1000$	$\nu = 10^{-4}$ $T = 30$ $N = 10000$	$\nu = 10^{-5}$ $T = 20$ $N = 1000$
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 [66]	4096	813	813	0.0245	0.2051	0.1190	0.1982
TF-Net [24]	4096	428	428	0.0225	0.2253	0.1168	0.2268
ResNet [67]	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

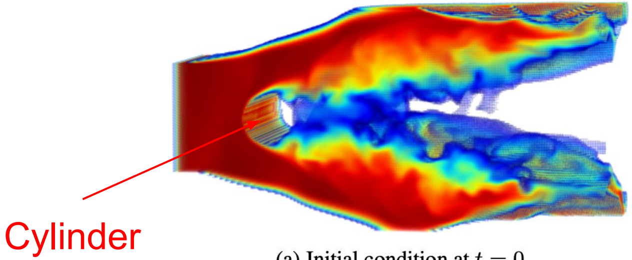
Our LE-PDE achieves competitive accuracy (compared to SOTA of FNO) while achieving significant speedup and using much less representation dimension.

Note that FNO-3D is not autoregressive, which directly map input to all output, and cannot extrapolate beyond the time range it is trained on. Therefore, for runtime, it is more fair to compare LE-PDE with FNO-2D (both autoregressive)

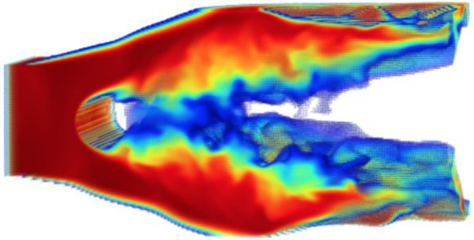
3.3. 3D Navier-Stokes Equation

$$\partial_t u_x + \mathbf{u} \cdot \nabla u_x = -\frac{1}{\rho}\nabla p + \nu \nabla \cdot \nabla u_x,$$
$$\partial_t u_y + \mathbf{u} \cdot \nabla u_y = -\frac{1}{\rho}\nabla p + \nu \nabla \cdot \nabla u_y,$$
$$\partial_t u_z + \mathbf{u} \cdot \nabla u_z = -\frac{1}{\rho}\nabla p + \nu \nabla \cdot \nabla u_z,$$
$$\text{subject to } \nabla \cdot \mathbf{u} = 0.$$

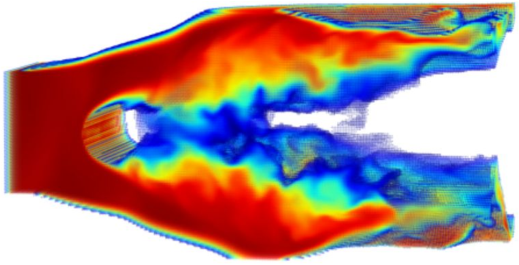
3D Navier-Stokes Equation (flow through the cylinder):



(a) Initial condition at $t = 0$

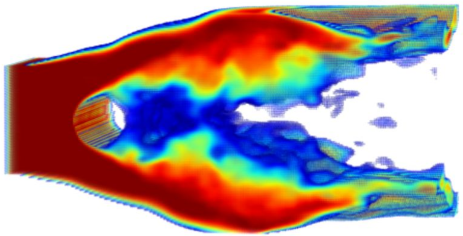


(c) Ground-truth at $t = 20$

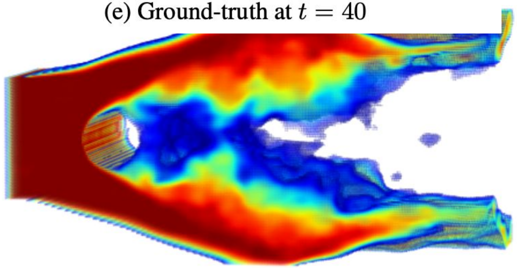


(e) Ground-truth at $t = 40$

LE-PDE prediction:



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



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

3D space is discretized into 256 x 128 x 128 grid, resulting in 4.19 million cells per time step

3.3. 3D Navier-Stokes Equation

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

(=256 x 128 x 128 x 4 features)

LE-PDE:

- 840× speedup compared to the ground-truth solver, and 12.3× speedup compared to the ablation model without latent evolution
- Reduce representation dimension by 130,000-fold

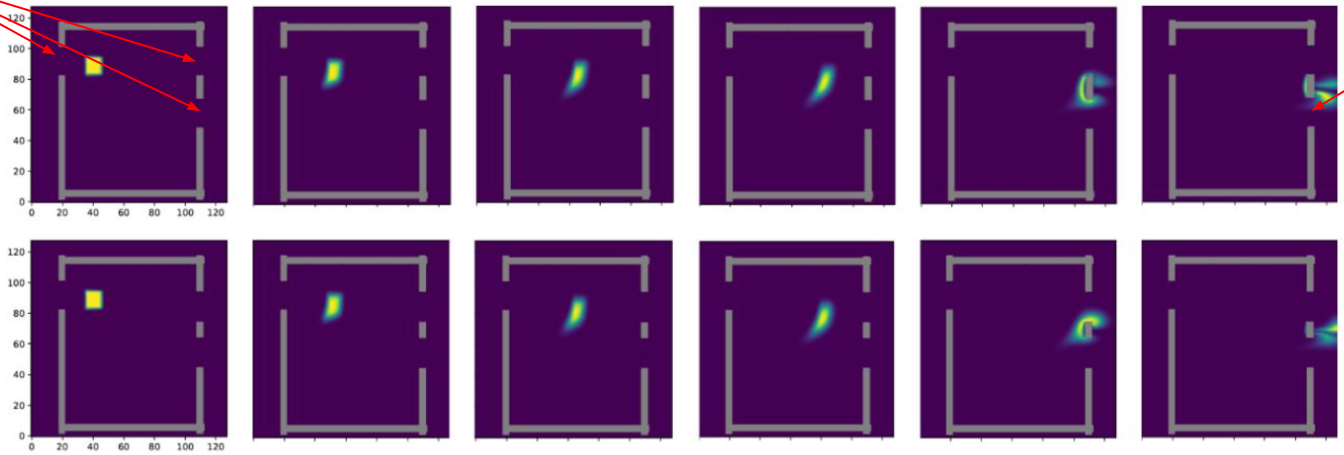
3.4. Inverse optimization of boundary

Optimize inlet and outlet location

T=0 T=15 T=30 T=45 T=60 T=70

Initial Random Boundary

Optimized Boundary with LE-PDE



Objective: make the fraction of smoke passing through lower outlet 30%

Task: The objective is to let the total amount of smoke pass through the lower outlet be 30%

This problem is challenging because:

- Objective depends on long-term rollout
- Objective very sensitive to boundary configuration
- Boundary mask is True/False, gradient cannot pass through

3.4. Inverse optimization of boundary

We compare our model with state-of-the-art learning-based model Fourier Neural Operator (FNO), and an ablation of without latent evolution (LE-PDE-~~latent~~):

Table 3: Comparison of LE-PDE with baselines. LE-PDE achieves above 1.7× speedup and much lower error computed by ground-truth solver.

	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**

Summary

We have introduced LE-PDE for forward simulation and inverse optimization of PDEs. Compared to state-of-the-art deep learning-based models, our LE-PDE:

- significantly speeds up, by up to **15x in speed**
- with **7.8x to 130,000x compression** in representation, compared to evolving it in input space (the larger system size the more compression)
- while achieving **competitive accuracy**

We are also looking for collaborators for further developments

For more, see our paper and project page at
http://snap.stanford.edu/le_pde/, or SCAN the QR code: