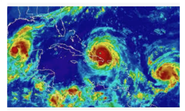
# Learning to Accelerate PDEs via Latent Global Evolution

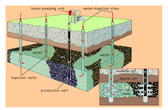
Tailin Wu[1], Takashi Maruyama[12], Jure Leskovec[1]  [1]Stanford University, [2]NEC Corporation

## Motivation

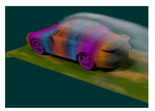
Partial differential equations (PDEs) are critical in science and engineering.
Two major tasks:

- Forward simulation: predicting the time-evolution of the system
- Inverse optimization: optimizing the boundary, etc. given forward model


Weather forecasting


Subsurface fluid simulation for oil production


Aerodynamics for Rocket


Equipment manufacturing

Key challenge: slow to simulate

- Due to large-scale in size: state dimension > millions per time step
- Requiring up to High Performance Computing (HPC)

## Existing simulators

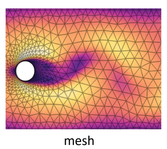
Prior methods and their challenges:

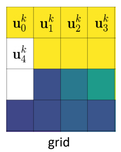
### Classical solvers

Based on Partial Differential Equations (PDEs), discretize the PDE, then use finite difference, finite element, etc. to evolve the system.

$$\frac{\partial \mathbf{u}}{\partial t} = F(x, \mathbf{u}, \frac{\partial \mathbf{u}}{\partial \mathbf{x}}, \frac{\partial^2 \mathbf{u}}{\partial \mathbf{x}^2}, ...) \qquad \mathbf{u}(t, \mathbf{x}) \longrightarrow \mathbf{u}_i^k \longrightarrow \text{discrete time index, discrete cell id}$$

Pros: (1) First principle-based, (2) accurate, (3) have error guarantee.
Challenges: Slow (computation scales with # cells)


mesh



grid

### Deep learning-based surrogate models

Pros: (1)Directly learn from data, alleviating much engineering efforts.
(2) Offer speedup via larger spatial/temporal intervals and explicit forward

Challenge: still 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)

### Reduced-order modeling:

Prior methods have limitations in their expressivity (e.g. linear projection to basis functions), generality (e.g. specific to certain classes of problems), or degradation of accuracy.

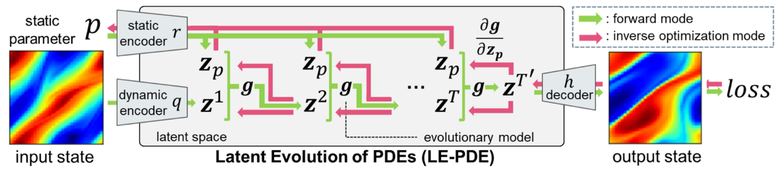
QR code for paper

QR code for open-source code

## Approach

Main idea: Learn a model that evolves the state in a global latent space, with a learning objective for accurate long-term prediction in latent and input space.

We introduce LE-PDE, a simple, fast and scalable method to accelerate the forward simulation and inverse optimization of PDEs



Latent Evolution of PDEs (LE-PDE)

We map an input state into a latent space (encoder: $q$) and learn a model $g$ that can evolve the state in the latent space instead of learning the evolution in the input space. And only when it needed, we decode it back into the original input space (decoder $h$) .

### Learning objective

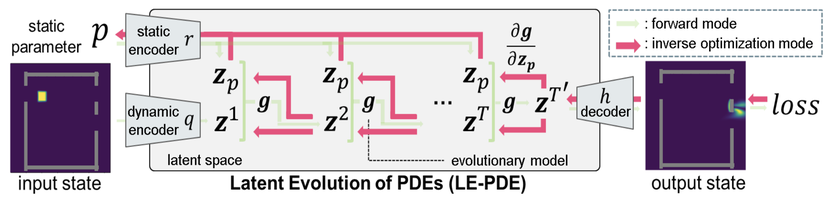
$$L = \frac{1}{K}\sum_{k=1}^{K}(L^k_{\text{multi-step}} + L^k_{\text{recons}} + L^k_{\text{consistency}}).$$

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

(consistent long-term prediction in latent space)

Inverse mode: use gradient descent to optimize static parameter (e.g. boundary), through the unrolled latent evolution:

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

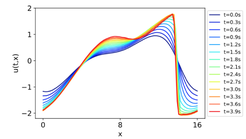


Latent Evolution of PDEs (LE-PDE)

## Result

LE-PDE achieves up to 15x speedup w.r.t. state-of-the-art deep learning-based surrogate models with competitive accuracy.

### PDE simulation: 1D


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

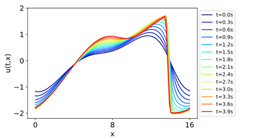
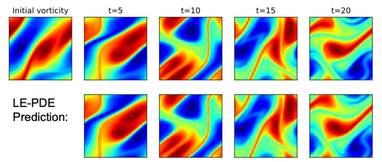

(d) Ground-truth with $n_x = 100$

Table 1: Performance of models in 1D for scenarios **E1**,**E2**, **E3**. Accumulated error $= \frac{1}{n_x}\sum_{t,x}$ MSE.

| | | 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 | 1.13 | $1.9 \times 10^3$ | 90 | 20 | **8** | 2500 | **128** |
| **E1** | (250, 50) | 6.23 | 29.98 | **0.51** | 1.67 | 1.20 | $1.8 \times 10^3$ | 80 | 20 | **8** | 1250 | **128** |
| **E1** | (250, 40) | 9.63 | 10.44 | **0.57** | 1.47 | 1.17 | $1.7 \times 10^3$ | 80 | 20 | **8** | 1000 | **128** |
| **E2** | (250, 100) | 1.19 | 17.09 | 2.53 | 1.58 | **0.77** | $1.9 \times 10^3$ | 90 | 20 | **8** | 2500 | **128** |
| **E2** | (250, 50) | 5.35 | 3.57 | 2.27 | 1.63 | **1.13** | $1.8 \times 10^3$ | 90 | 20 | **8** | 1250 | **128** |
| **E2** | (250, 40) | 8.05 | 3.26 | 2.38 | 1.45 | **1.03** | $1.7 \times 10^3$ | 80 | 20 | **8** | 1000 | **128** |
| **E3** | (250, 100) | 4.71 | 10.16 | 5.69 | 4.26 | **3.39** | $4.8 \times 10^3$ | 90 | 19 | **6** | 2500 | **64** |
| **E3** | (250, 50) | 11.71 | 14.49 | 5.39 | **3.74** | 3.82 | $4.5 \times 10^3$ | 90 | 19 | **6** | 1250 | **64** |
| **E3** | (250, 40) | 15.94 | 20.90 | 5.98 | **3.70** | 3.78 | $4.4 \times 10^3$ | 90 | 20 | **8** | 1000 | **128** |

### PDE simulation: 2D




### PDE simulation: 3D (4 million cells/step)

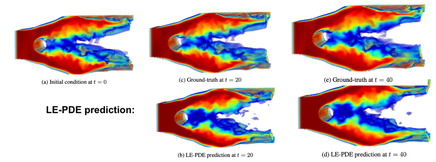



Table 2: Performance of different models in 2D Navier-Stokes flow. Runtime is using the $\nu = 10^{-3}, N = 1000$ for predicting 40 steps in the future.

| 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 |

### Inverse optimization

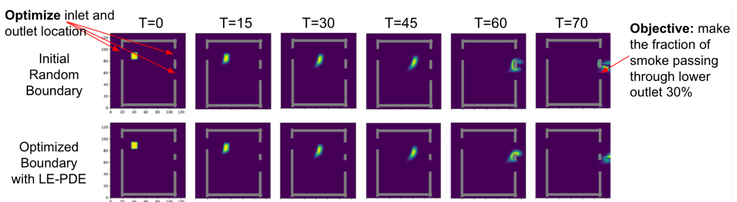
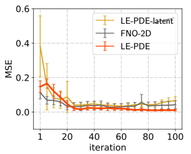
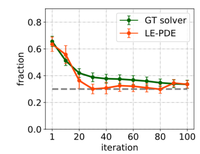



Table 3: Comparison of LE-PDE with baselines.

| | 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** |



(d) Objective function.



(e) Smoke amount at lower outlet.