Microsoft

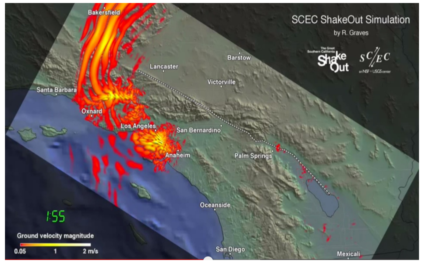
PDE-Refiner: Achieving Accurate Long Rollouts with Neural PDE Solvers

Phillip Lippe[1,2], Bas S. Veeling[1], Paris Perdikaris[1], Richard E. Turner[1], Johannes Brandstetter[1]

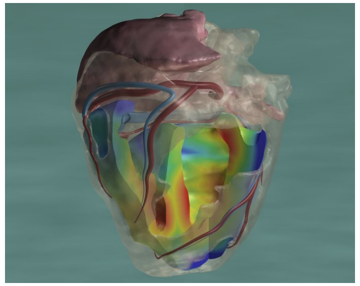
[1]Microsoft Research AI4Science, [2]University of Amsterdam

Project Website

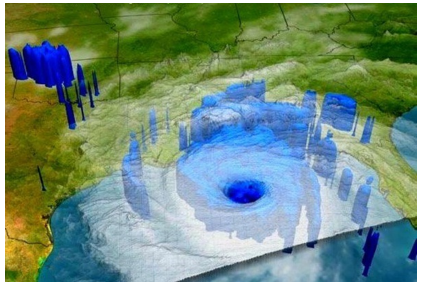
(Large-scale) PDE systems are ubiquitous



Earthquakes

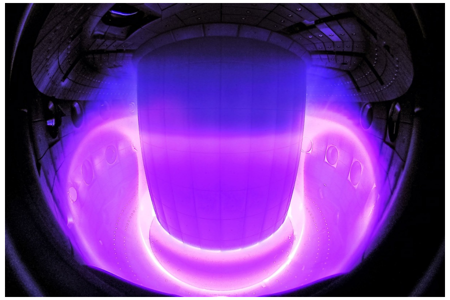


Heart dynamics

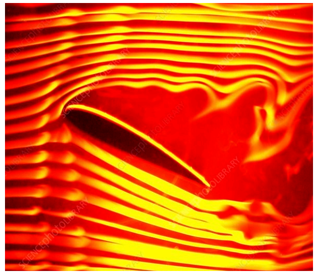


Weather prediction

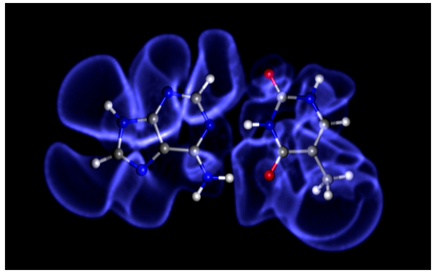
Galaxy collisions



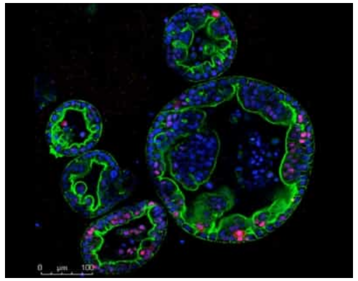
Plasma physics



Airplane design



Electronic structure



Tumor growth

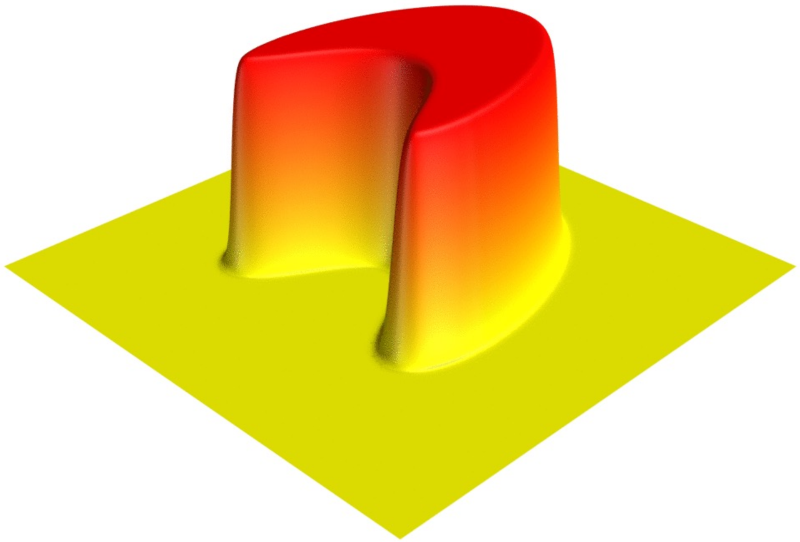
Solving PDEs – the traditional way

- Formulation of time dependent Partial Differential Equations (PDEs):

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

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

- Partition spatial and temporal domain into grid
- Estimate spatial derivatives, e.g., via finite difference
- Solve time derivative with classical ODE solvers, e.g., Runge-Kutta methods



Example: Heat Equation (credit)

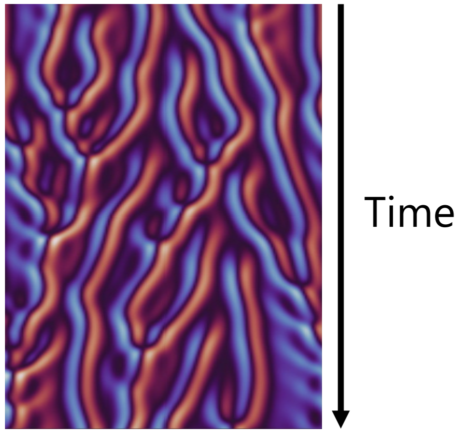
Example PDEs

- 2D Kolmogorov Flow (KM)
  - Fluid Dynamics
  - Incompressible Navier-Stokes
  - Known for its chaotic behavior
  - Accurate solving requires expensive, high resolution

$$\partial_t \mathbf{u} + \nabla \cdot (\mathbf{u} \otimes \mathbf{u}) = \nu \nabla^2 \mathbf{u} - \frac{1}{\rho} \nabla p + \mathbf{f}$$

- 1D Kuramoto-Sivashinsky Equation (KS)
  - Fourth-order nonlinear PDE
  - Fluid Dynamics, e.g., plasmas and flame propagation
  - Rich dynamical characteristics and chaotic behavior

$$u_t + uu_x + u_{xx} + \nu u_{xxxx} = 0.$$



Challenges for classical solvers

- Small errors have large long-term impact in chaotic PDEs
- Requires small time steps and/or large resolution ⇒ expensive

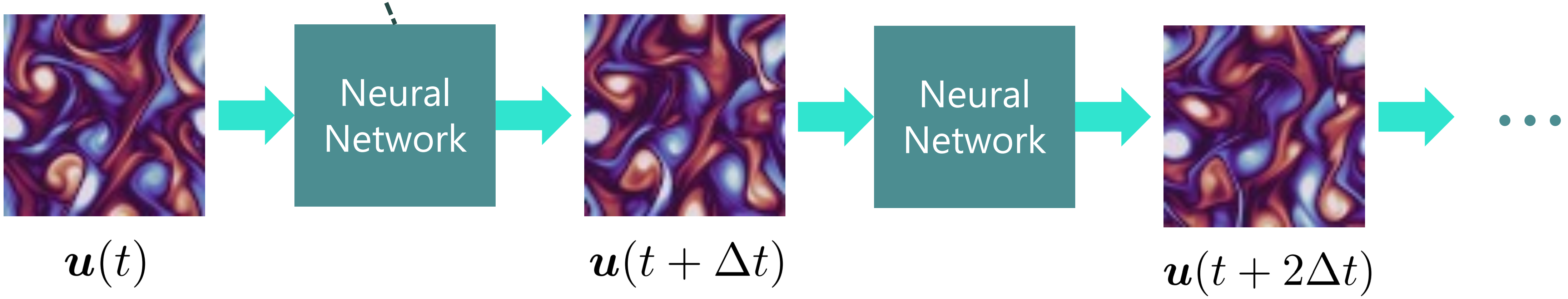
- Example: Kolmogorov Flow
  - Requires resolution of 2048x2048
  - Time step of 0.007s
  - 20 second rollout takes >30 minutes on an A100

- Can we use ML to solve PDEs more efficiently?

Neural PDE Solvers

- Neural Operators learn to predict future solutions

$$\boldsymbol{u}(t+\Delta t) = \mathcal{G}_t(\Delta t, \boldsymbol{u}(t))$$



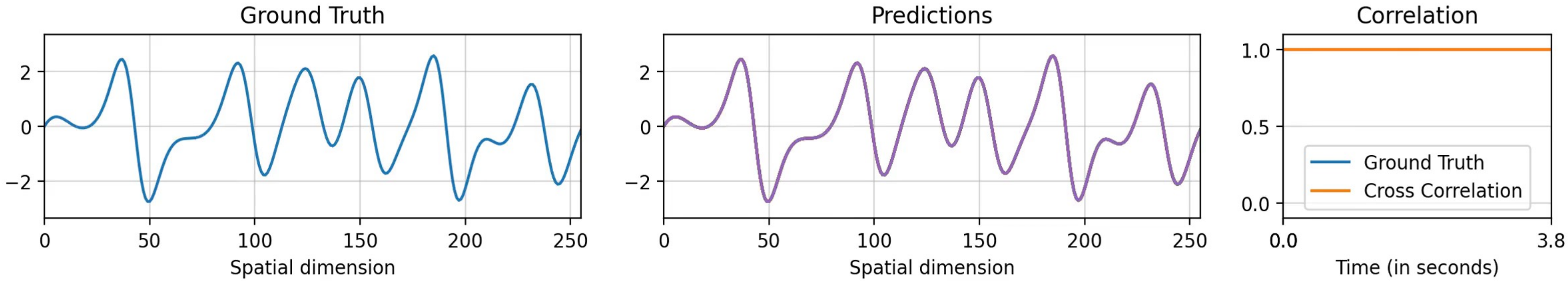
- Trained on one-step predictions
- Long horizon predictions via autoregressive rollout

Neural PDE Solvers - Desiderata

1. Long-Horizon Accuracy
   - Remain close to ground truth solution for long time
2. Long-Horizon Stability
   - Generate physically realistic solutions and not diverge
3. Uncertainty Estimation
   - Know when not to trust your neural surrogate anymore

Neural PDE Solvers - Desiderata

1. Long-Horizon Accuracy
   - Remain close to ground truth solution for long time
2. Long-Horizon Stability
   - Generate physically realistic solutions and not diverge
3. Uncertainty Estimation
   - Know when not to trust your neural surrogate anymore



Challenges of Accurate Long Rollouts

Training Neural PDE Solvers

- Commonly trained with one-step MSE:

$$\mathcal{L}_{\text{MSE}} = \|u(t) - \texttt{NO}(u(t - \Delta t))\|^2$$

- Tradeoff in time step size
  - Large time steps give fast solvers, but harder to learn
  - Small time steps are easier to learn and generalize, but require many autoregressive steps
  - In practice, small time steps commonly achieve better performance

- Evaluate on rolling out model on its own predictions
  - Check when it diverges from ground truth, e.g., in terms of correlation of MSE loss

Errors during Autoregressive Rollout

Three main sources of error during rollout:

1. Error Accumulation

- Models estimate the temporal difference between time steps, added to the original input
- Errors on the initial input are forwarded to future steps during rollout

2. Error Propagation

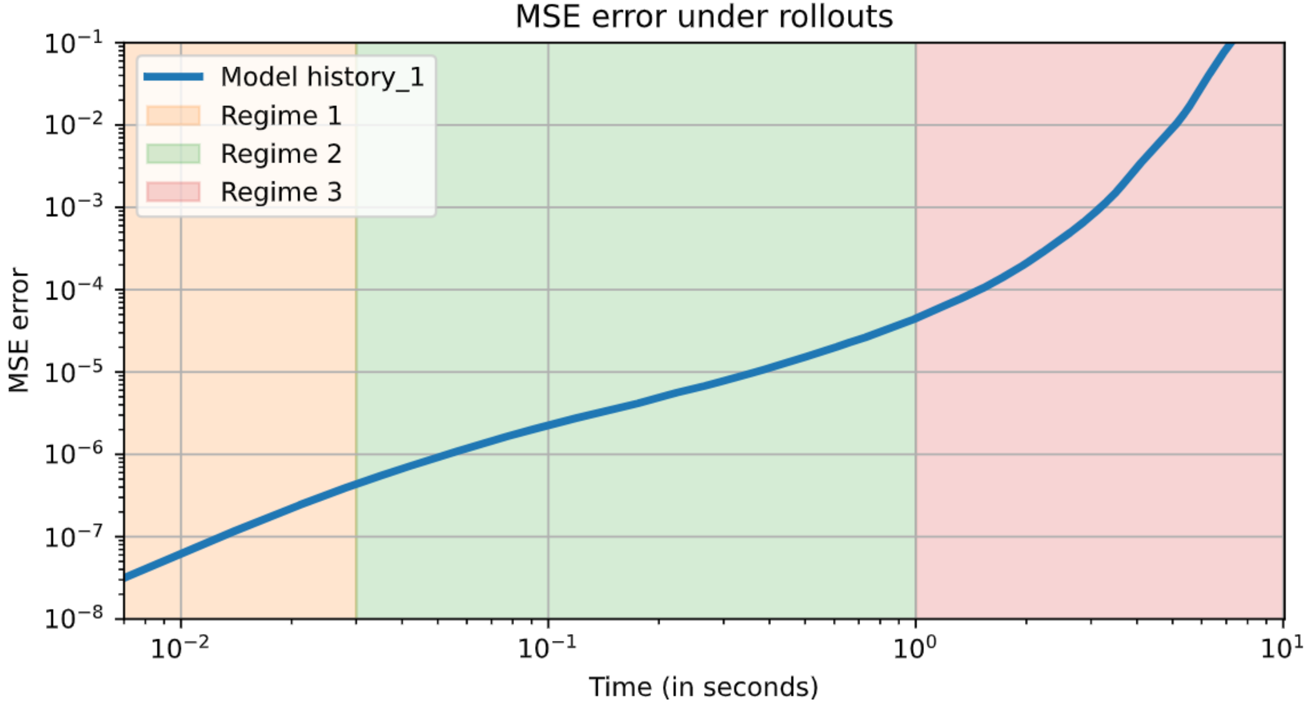
- Errors on network input influence the predicted temporal difference / dynamics

3. Input Distribution Shift

- Models are trained on ground truth data
- Predictions with errors may have a different distributions
- Can cause the model to diverge and make arbitrary predictions

Autoregressive Error Propagation

2D Kolmogorov Flow



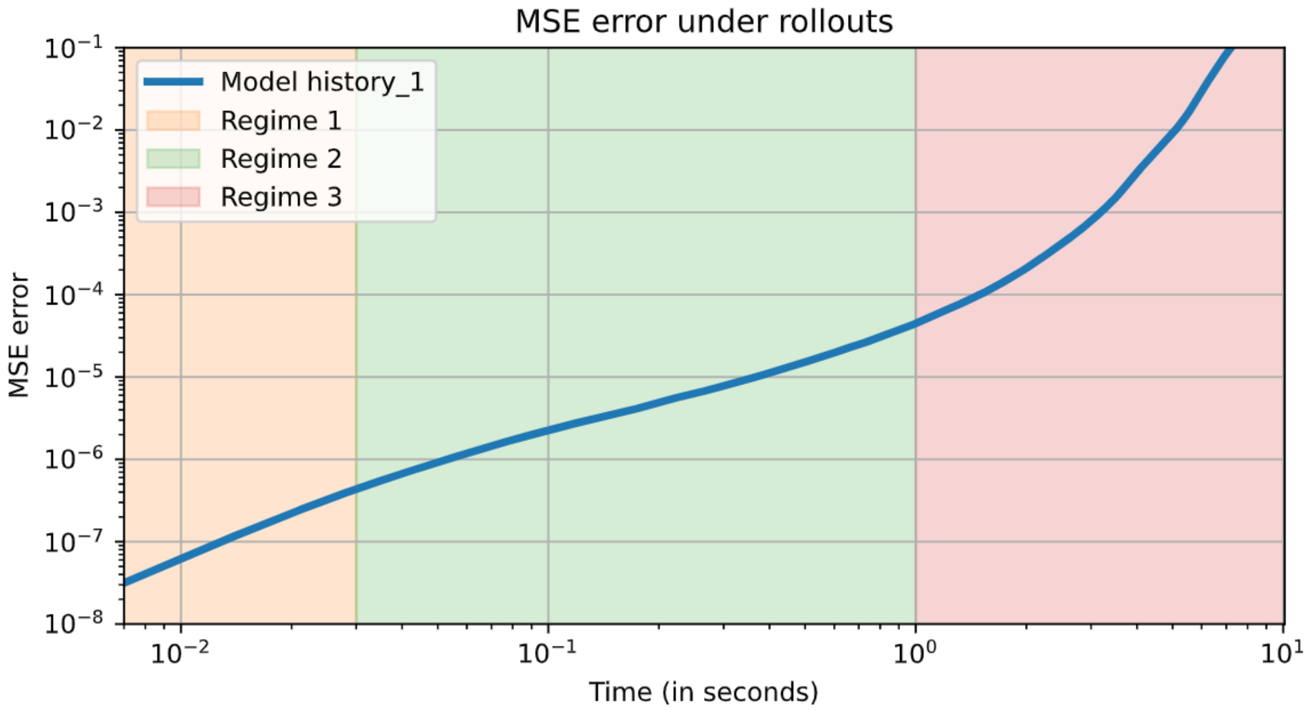
Regime 1

- Only error accumulation
- Consecutive errors are highly correlated ⇒ quadratic increase of error

$$\varepsilon_t \sim t^2 \varepsilon_0$$

Autoregressive Error Propagation

2D Kolmogorov Flow

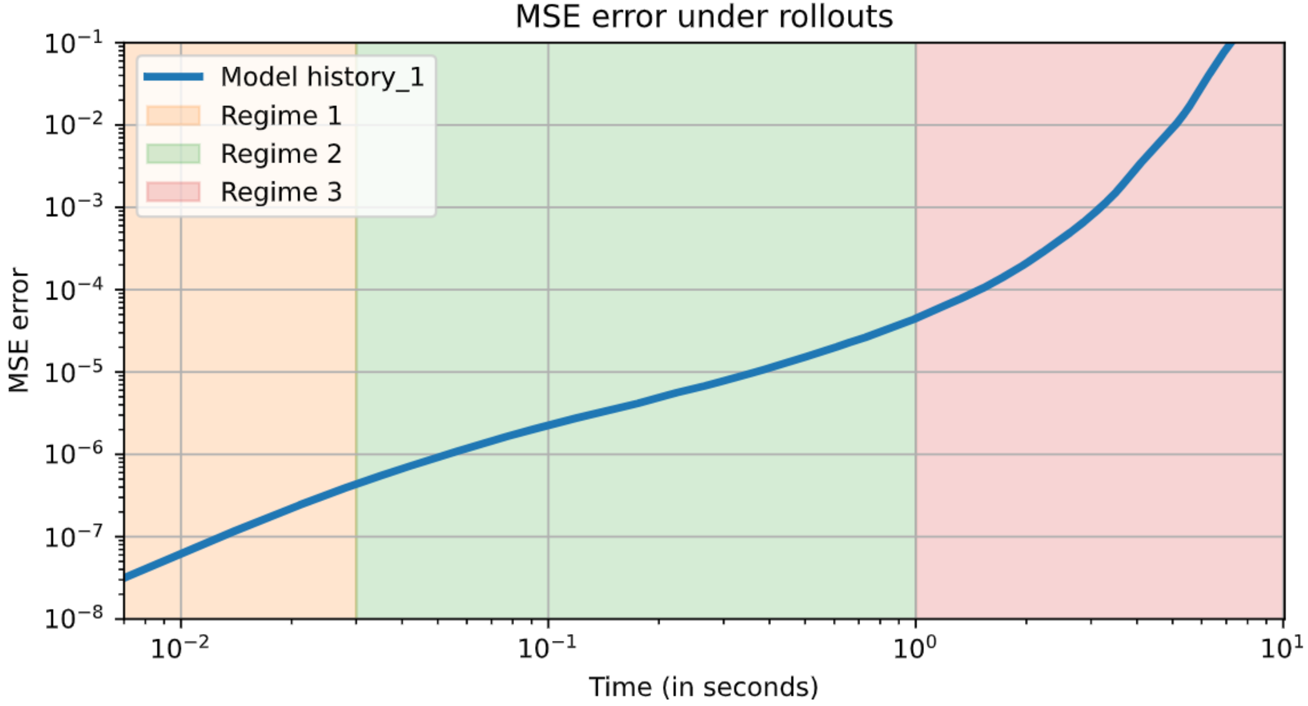


Regime 2

- Mainly error accumulation
- Errors become less correlated
  $\Rightarrow$ slower increase of error
  $$\varepsilon_t \sim t^{1.5} \varepsilon_0$$

Autoregressive Error Propagation

2D Kolmogorov Flow

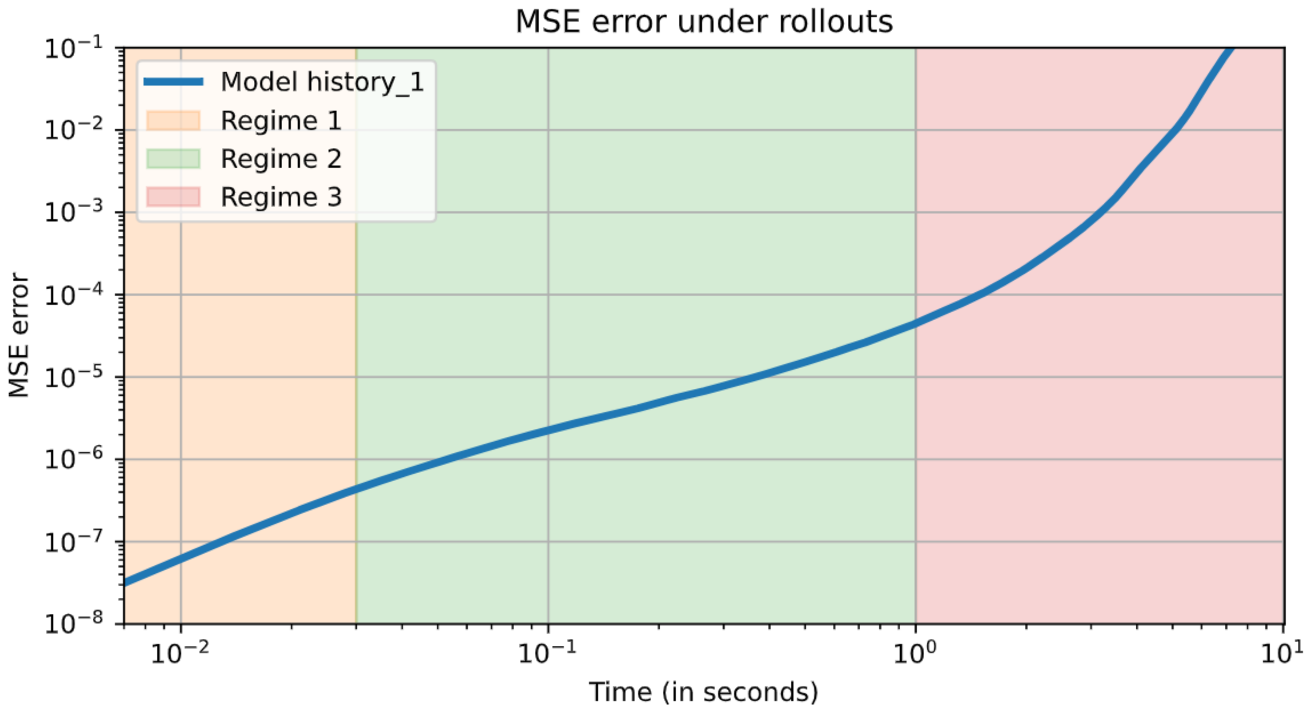


Regime 3

- Error propagation dominates
- Errors exponentially increase, diverge from ground truth
- Predictions yet appear physically realistic

Autoregressive Error Propagation

2D Kolmogorov Flow

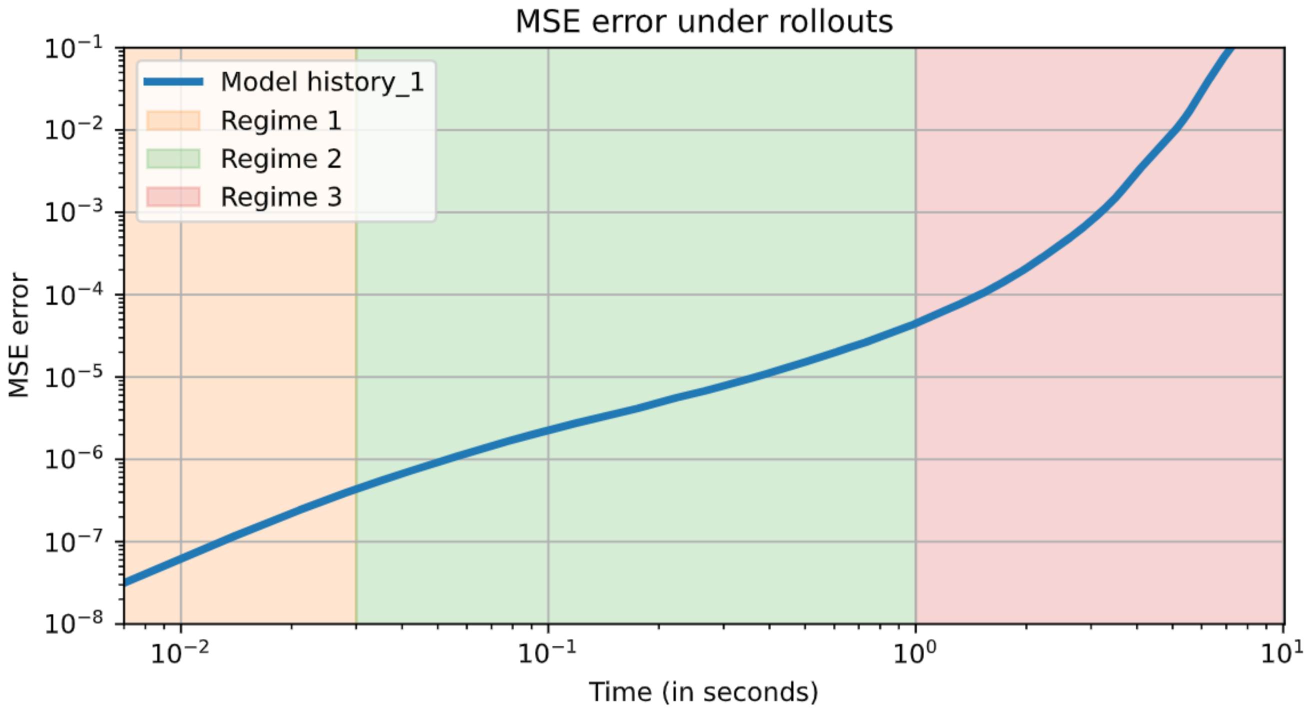


Regime 4

- Input Distribution Shift occurs much later in our setup
- Often after >10 times longer rollout than divergence time

Autoregressive Error Propagation

2D Kolmogorov Flow



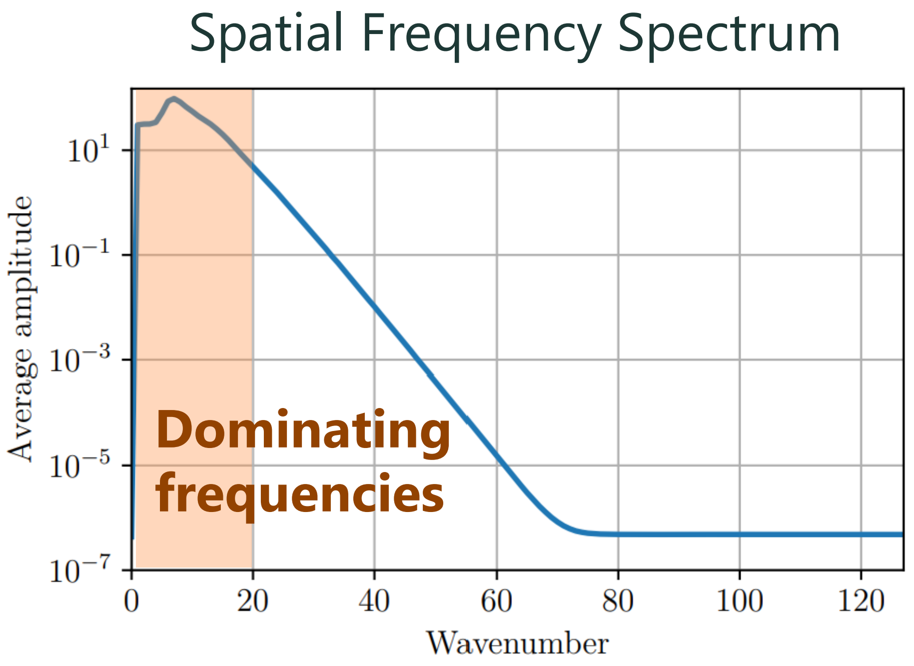
Rollout potentially improves by:

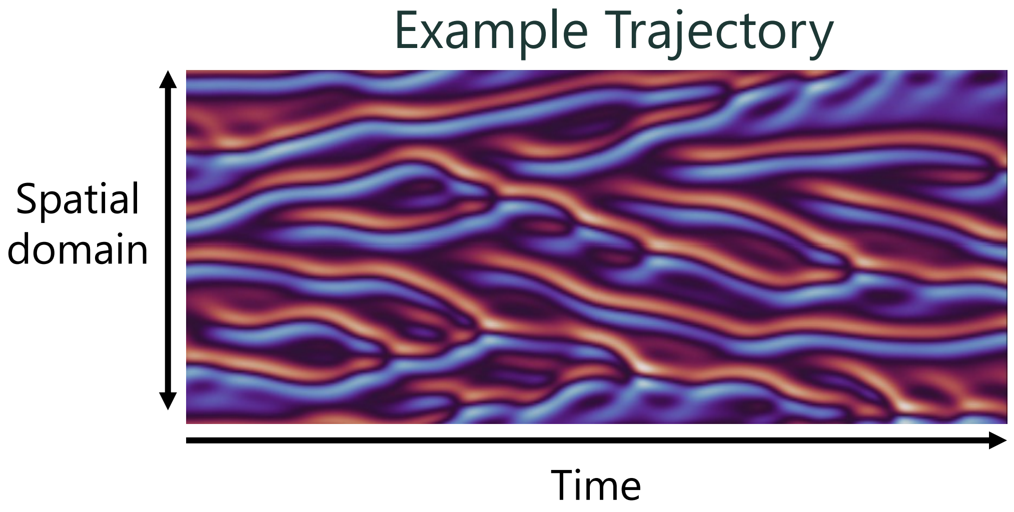
- Lower one-step loss
- Delaying error propagation (regime 3)

Delaying Error Propagation

- Example: 1D Kuramoto-Sivashinsky equation (KS)

$$u_t + uu_x + u_{xx} + \nu u_{xxxx} = 0$$





Delaying Error Propagation

- Example: 1D Kuramoto-Sivashinsky equation (KS)

$$u_t + \boxed{uu_x} + \boxed{u_{xx} + \nu u_{xxxx}} = 0.$$

Non-linear term causes all spatial frequencies to interact long-term

High-order derivatives increase importance of high frequencies in spatial domain

For long accurate rollouts, model **all** spatial frequencies accurately
Errors in higher frequencies have low short-term, but **high long-term impact**

Delaying Error Propagation

- Many challenging PDEs follow similar pattern, for example:

KS equation $u_t + \boxed{uu_x} + \boxed{u_{xx} + \nu u_{xxxx}} = 0$

Burger's equation $u_t + \boxed{uu_x} + \boxed{u_{xx}} = 0$

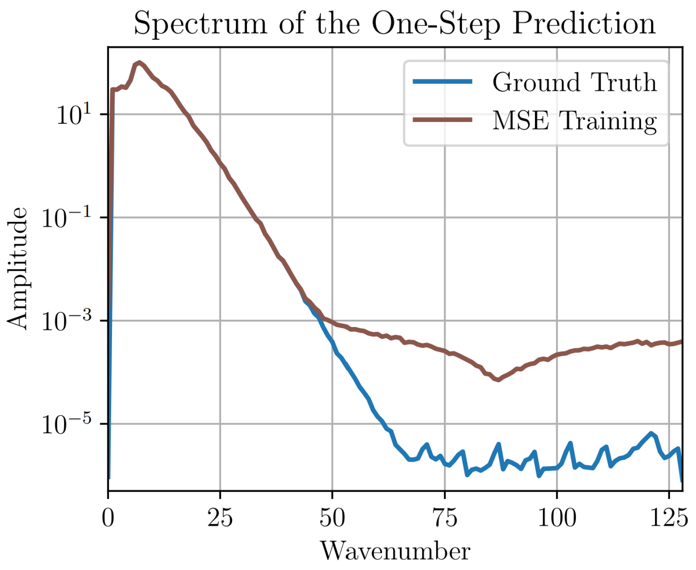
Korteweg-de Vries equation $u_t + \boxed{6uu_x} + \boxed{u_{xxx}} = 0$

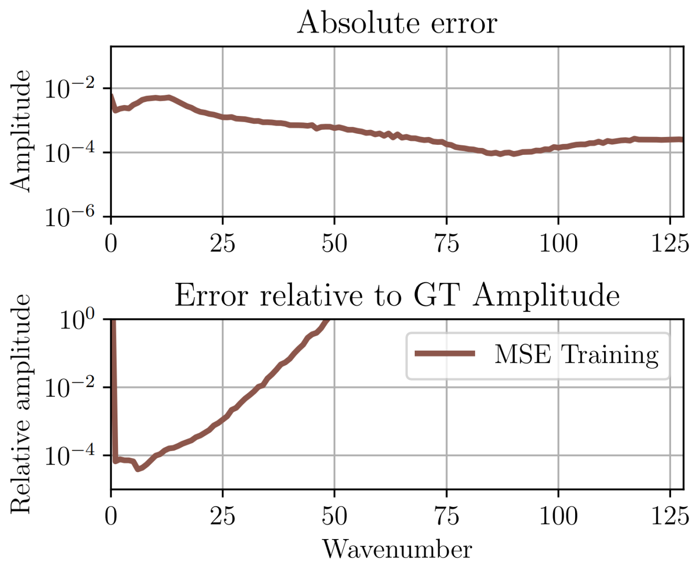
KdV-Burger's equation $u_t + \boxed{2uu_x} - \boxed{\nu u_{xx} + \mu u_{xxx}} = 0$

For long accurate rollouts, model **all** spatial frequencies accurately
Errors in higher frequencies have low short-term, but **high long-term impact**

Case Study: Kuramoto-Sivashinsky

- How well do MSE-trained surrogates cover the frequency spectrum?





- Neural surrogates focus on **dominating** frequencies, losing high frequencies
- Inherently limits the maximum rollout time

Challenges of Accurate Long Rollouts – Summary

- Main causes for divergence: error accumulation and error propagation
- History information improves one-step, but accelerates error propagation
- MSE surrogates poorly model low-amplitude frequencies, inevitably setting a maximum possible rollout time

How can we capture the **whole** frequency spectrum better?

PDE-Refiner

Achieving Accurate Long Rollouts via an Iterative Refinement Process

Idea

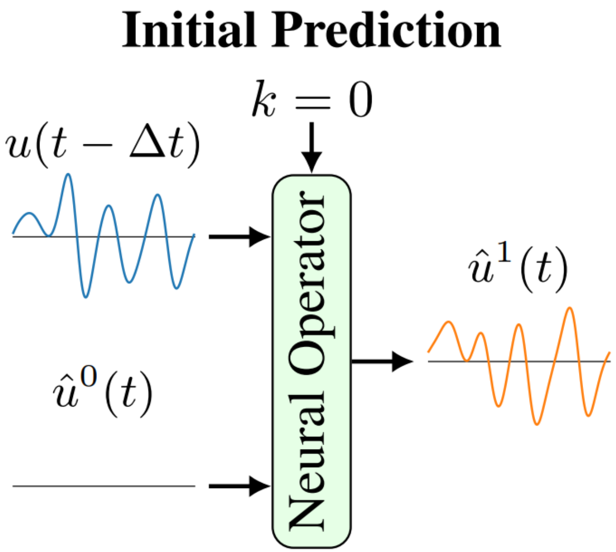
- Goal: improve prediction of low-amplitude frequencies

- Difficult to predict all frequencies perfectly at once
  ⇒ iterative refinement process to finetune the prediction step-by-step

- At each refinement step, focus on information/error below a certain amplitude
  - Implemented via a denoising objective

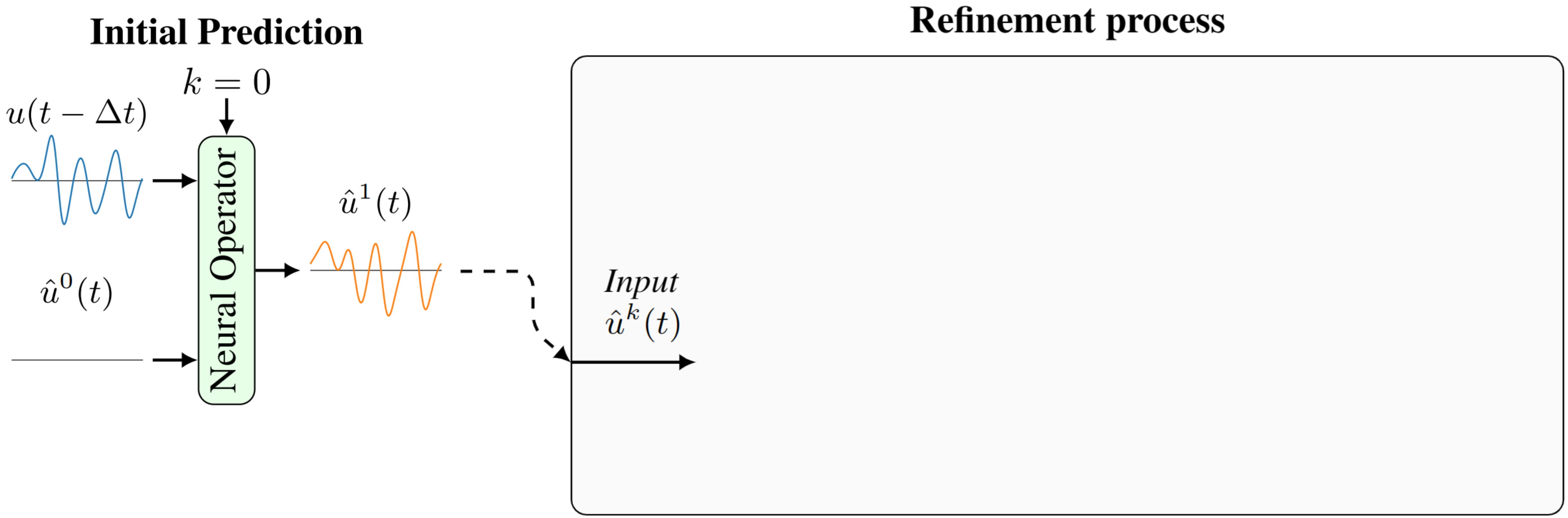
- Use multiple refinement steps to cover larger spectrum

PDE-Refiner



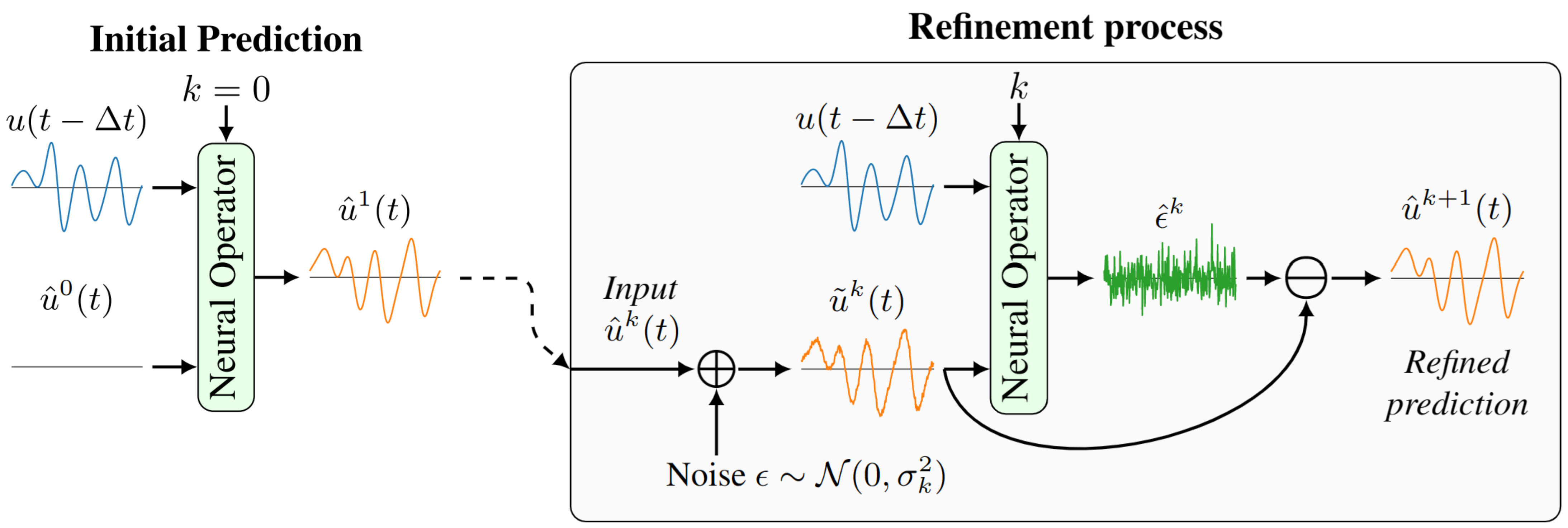
Initial Prediction
$k = 0$
$u(t - \Delta t)$
$\hat{u}^0(t)$
Neural Operator
$\hat{u}^1(t)$

PDE-Refiner



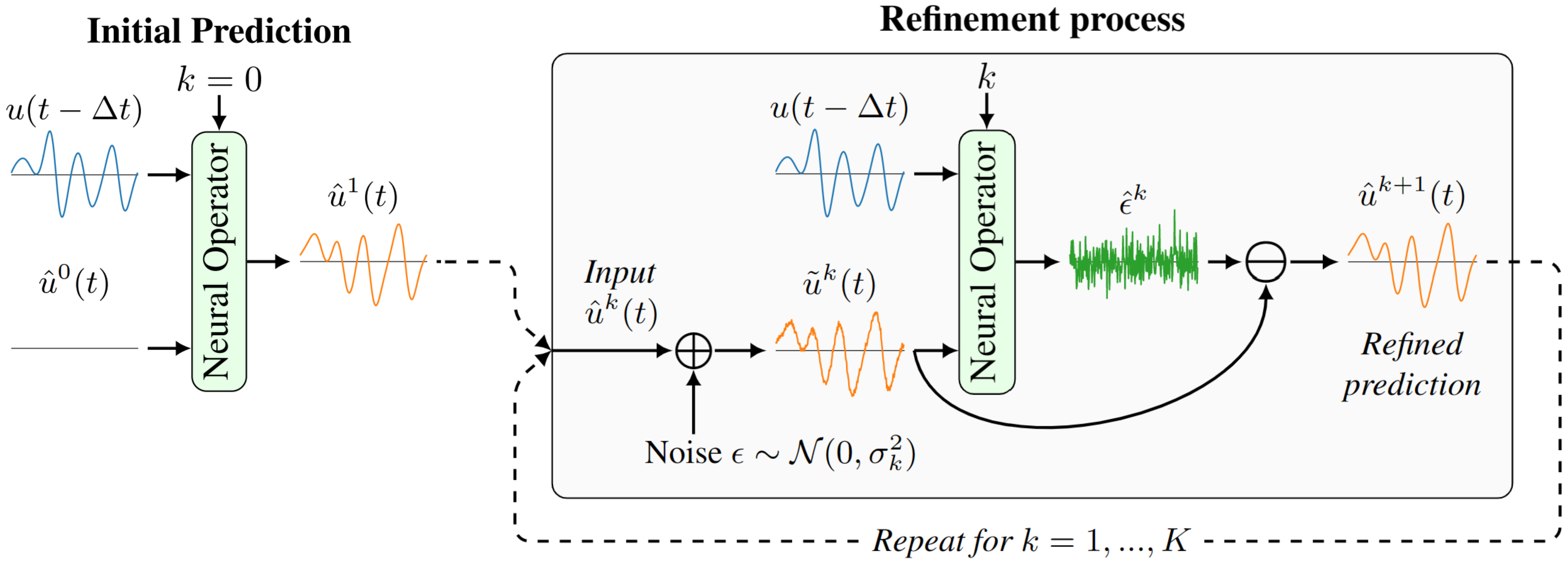
Initial Prediction
$k = 0$
$u(t - \Delta t)$
Neural Operator
$\hat{u}^1(t)$
$\hat{u}^0(t)$
Refinement process
Input
$\hat{u}^k(t)$

PDE-Refiner



Initial Prediction
$k = 0$
$u(t - \Delta t)$
Neural Operator
$\hat{u}^1(t)$
$\hat{u}^0(t)$
Refinement process
k
$u(t - \Delta t)$
Neural Operator
$\hat{\epsilon}^k$
$\hat{u}^{k+1}(t)$
Input
$\hat{u}^k(t)$
$\tilde{u}^k(t)$
Refined prediction
Noise $\epsilon \sim \mathcal{N}(0, \sigma_k^2)$

PDE-Refiner



Initial Prediction
$k = 0$
$u(t - \Delta t)$
Neural Operator
$\hat{u}^1(t)$
$\hat{u}^0(t)$
Refinement process
k
$u(t - \Delta t)$
Neural Operator
$\hat{\epsilon}^k$
$\hat{u}^{k+1}(t)$
Input
$\hat{u}^k(t)$
$\tilde{u}^k(t)$
Refined prediction
Noise $\epsilon \sim \mathcal{N}(0, \sigma_k^2)$
Repeat for $k = 1, \ldots, K$

PDE-Refiner – Training

- Initial prediction: common MSE objective

$$\mathcal{L}^0(u,t) = \|u(t) - \texttt{NO}\left(\underbrace{\hat{u}^0(t)}_{=0}, u(t-\Delta t), 0\right)\|_2^2$$

- Refinement steps: denoise ground truth data
  - Training on GT learns to model the data's frequency spectrum

$$\mathcal{L}^k(u,t) = \mathbb{E}_{\epsilon^k \sim \mathcal{N}(0,1)}\left[\|\epsilon_k - \texttt{NO}\left(u(t) + \sigma_k \epsilon_k, u(t-\Delta t), k\right)\|_2^2\right]$$

- Use exponential decreasing noise variance $\sigma_k = \sigma_{\min}^{k/K}$

PDE-Refiner – Relation to Diffusion Models

- Popular usage of denoising: Diffusion Models (DDPM) [Ho et al., 2020]

- Key differences of PDE-Refiner to DDPMs:
  1. GT is deterministic ⇒ exponential decreasing noise schedule with very small minimum
  2. Speed is of essence for application ⇒ very few denoising steps (usually 1-4)
  3. Different objective ⇒ predicts signal at initial step

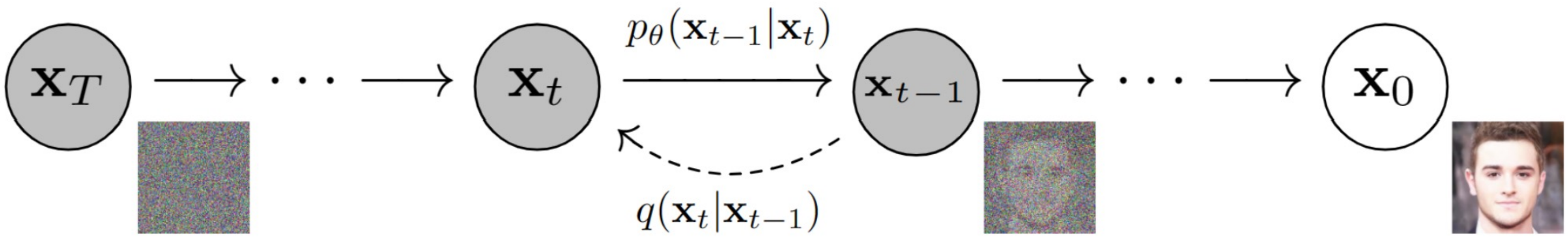
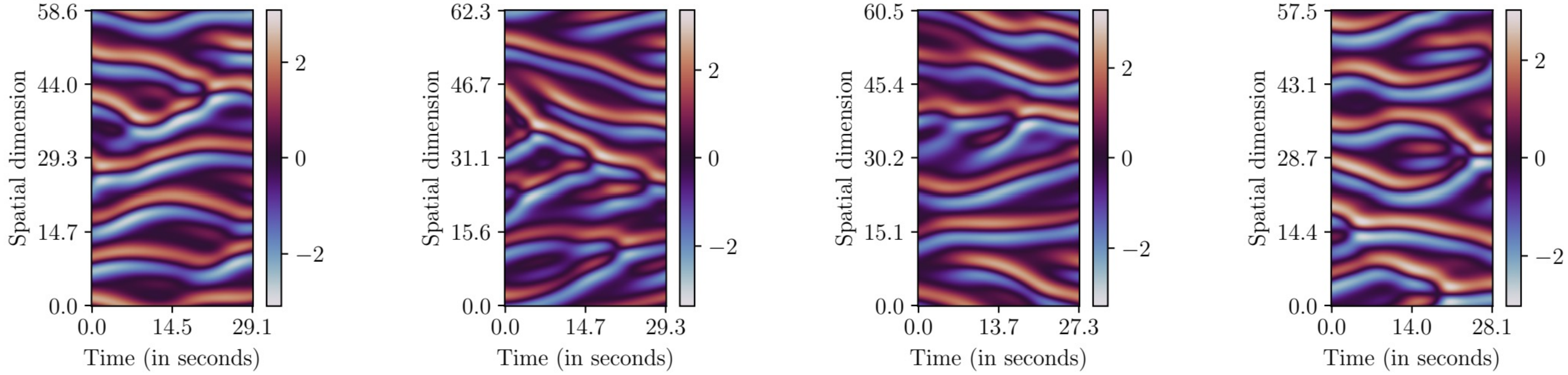


Figure credit: [Ho et al., 2020] Ho, J., Jain, A., & Abbeel, P. (2020). Denoising diffusion probabilistic models. *NeurIPS, 2020.*

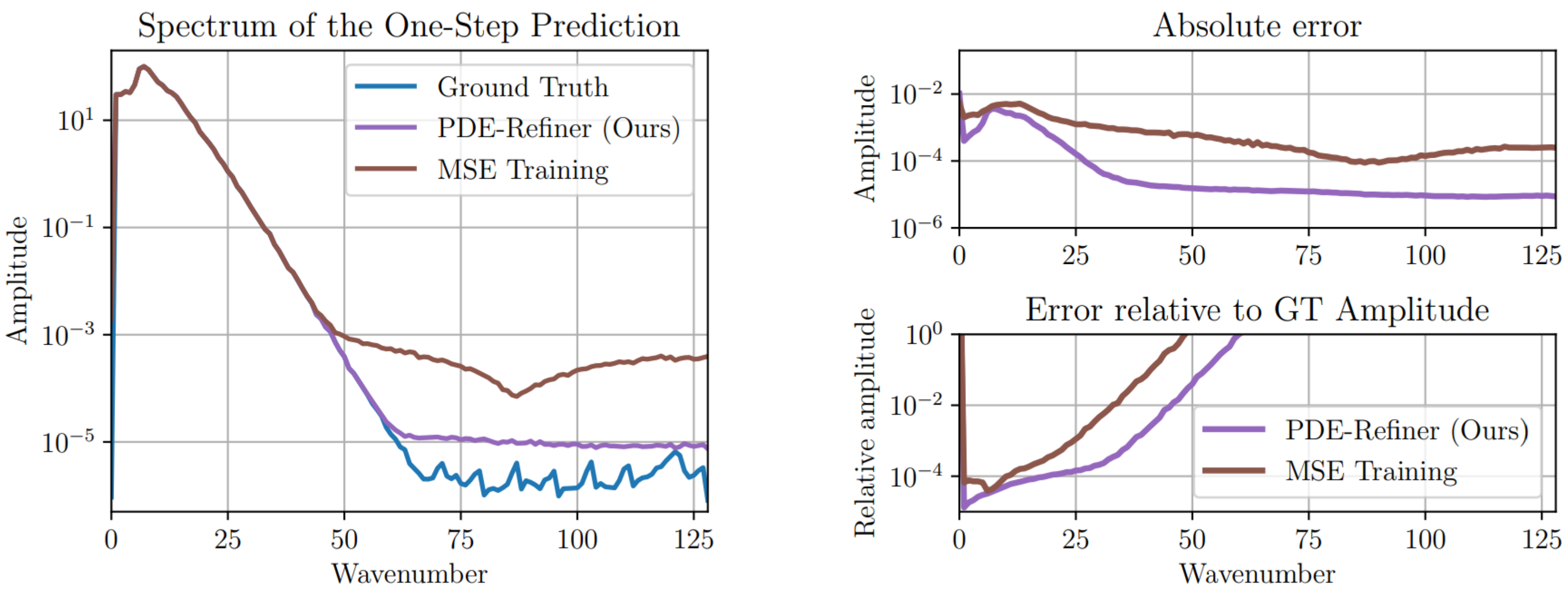
PDE-Refiner – Experimental Setup

- 1D Kuramoto-Sivashinsky Equation
  - Train models on simulated data from classical solver as ground truth
  - Varying initial condition, spatial size, and simulated time step
  - Different neural operator architectures (U-Net, FNO, Dilated ResNets)
  - Evaluation metric: time until correlation between rollout prediction and GT goes below 0.8



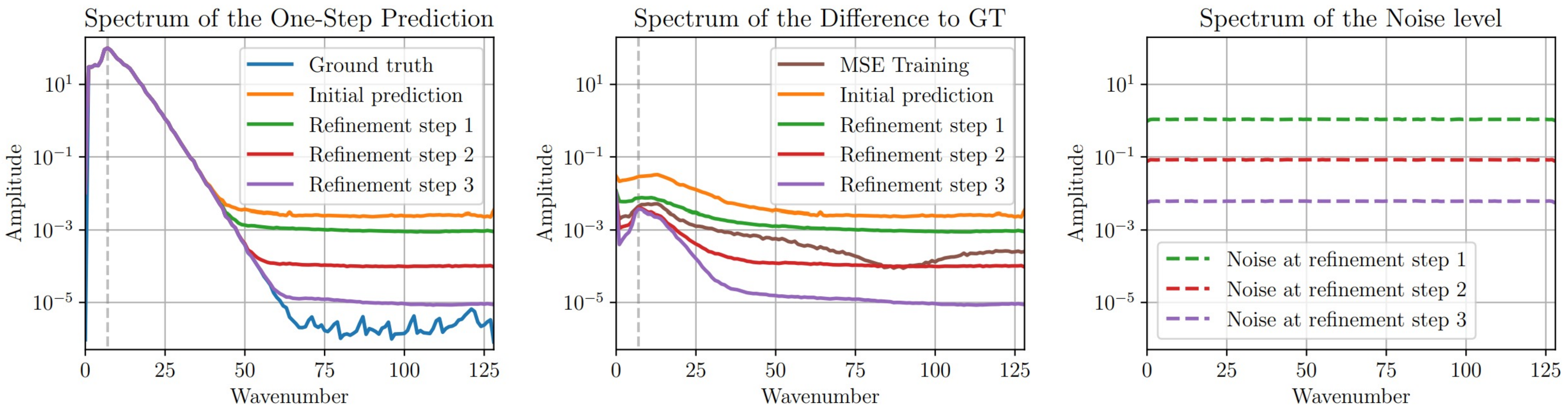
PDE-Refiner – Frequency Spectrum KS equation

- PDE-Refiner models a larger frequency band accurately

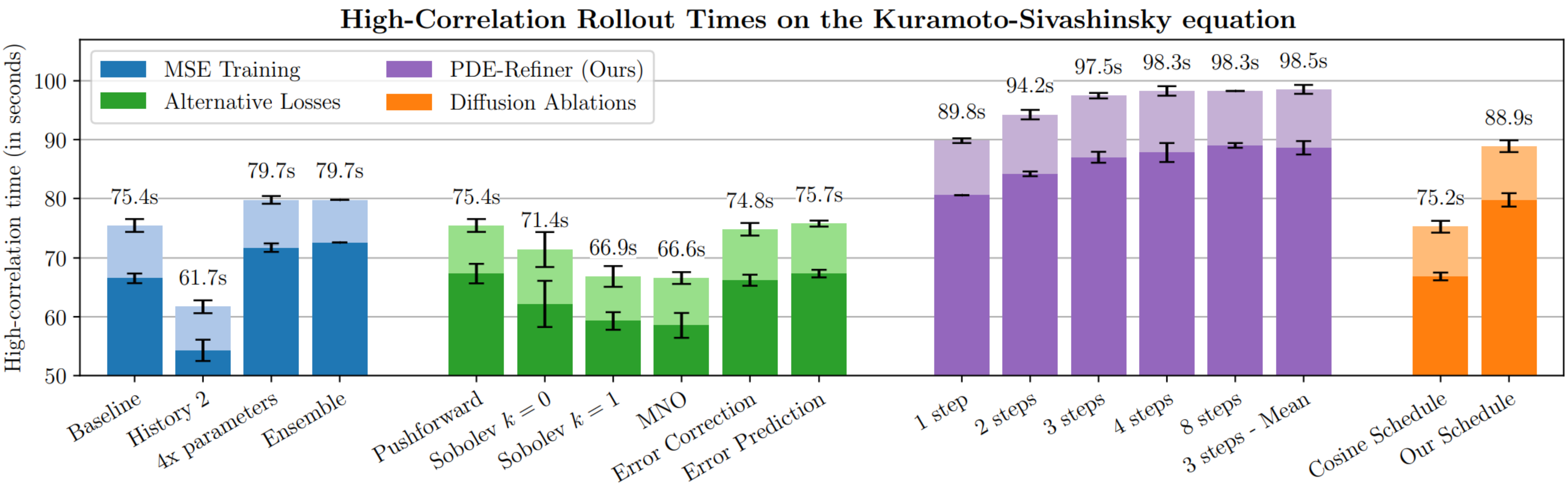


PDE-Refiner – Frequency Spectrum KS equation

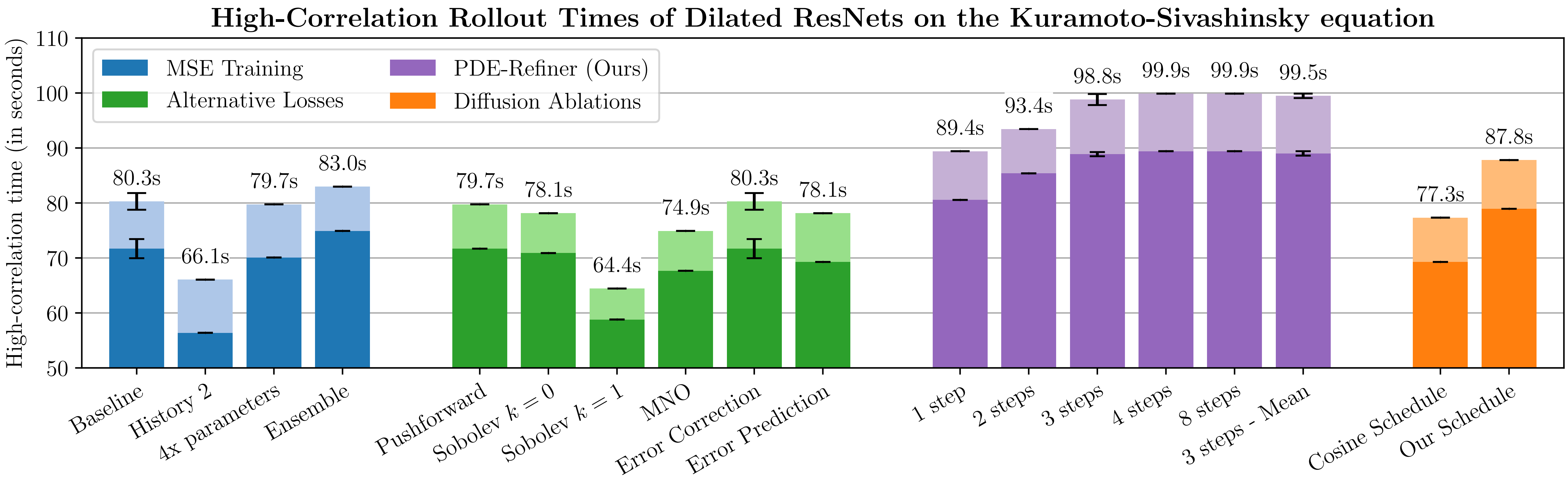
- Refinement steps focus on different amplitude levels



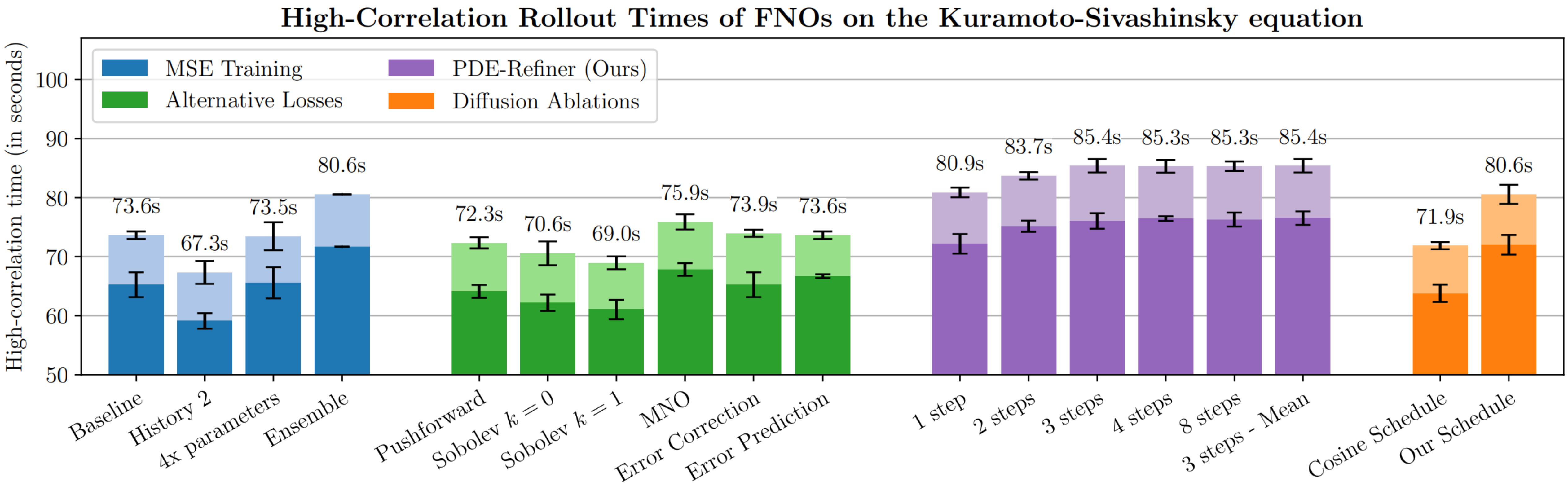
PDE-Refiner – Rollout Performance (U-Net)



PDE-Refiner – Rollout Performance (Dilated ResNets)

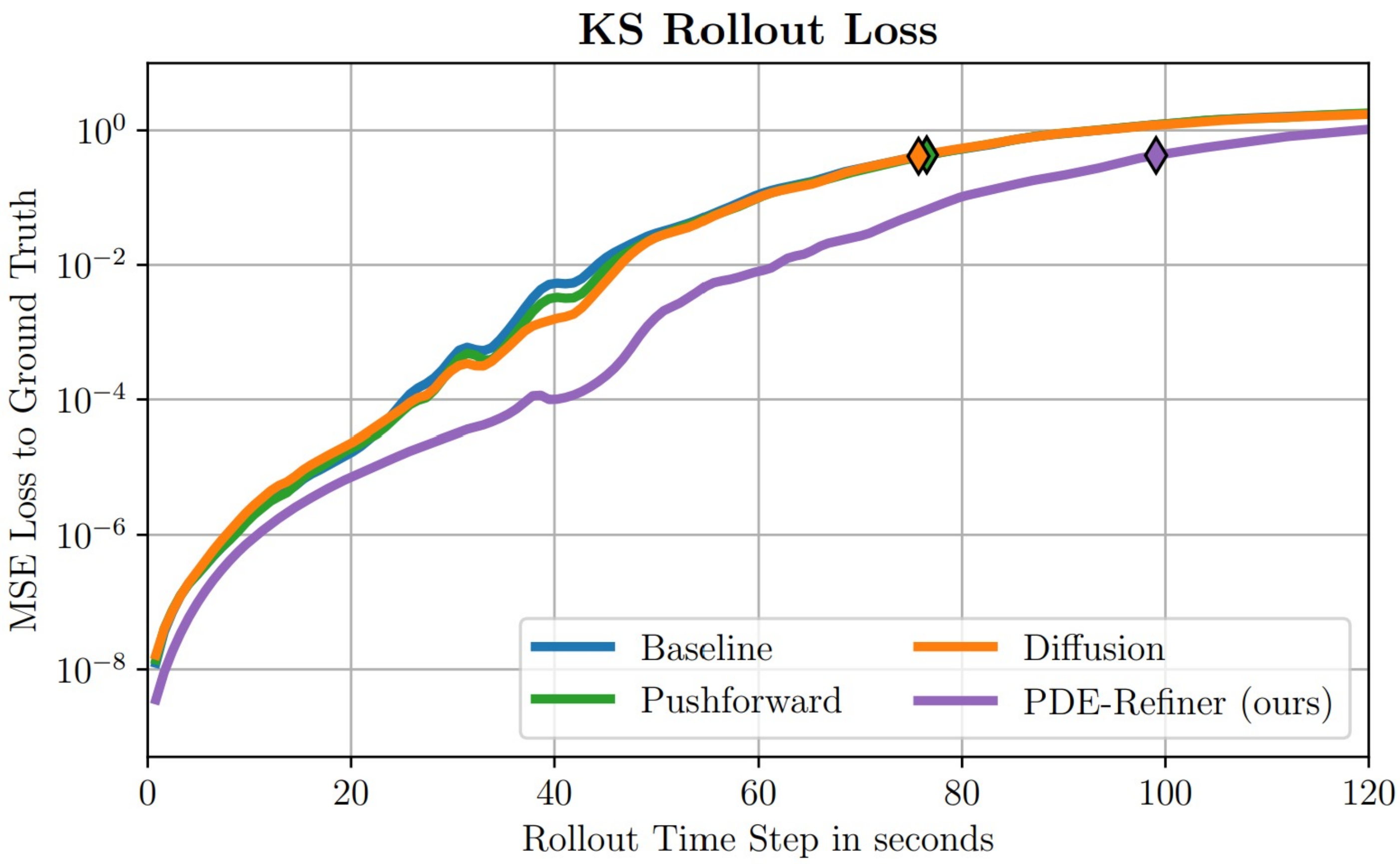


PDE-Refiner – Rollout Performance (FNOs)

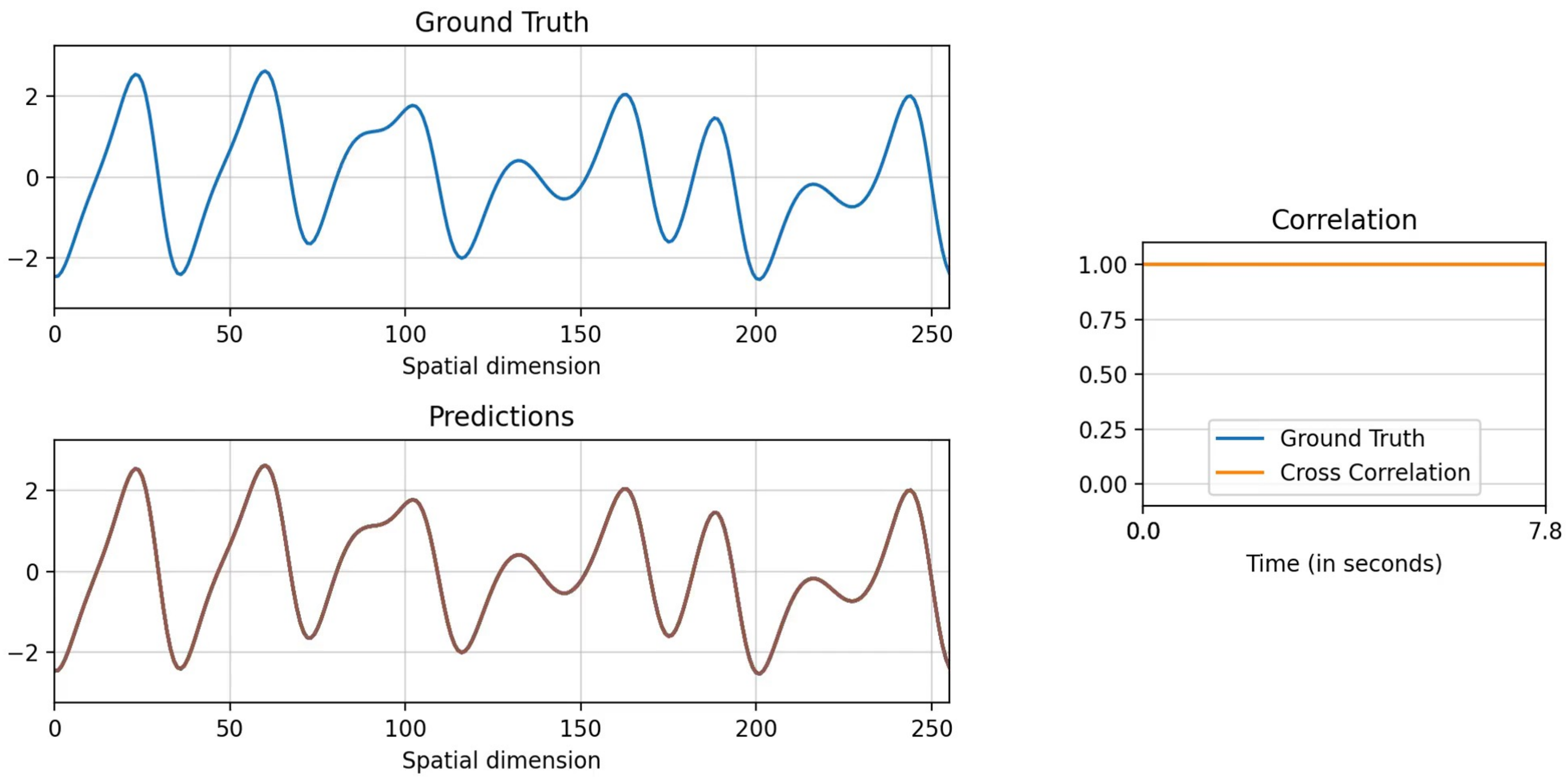


PDE-Refiner – Delaying Error Propagation

- MSE loss of all models similar for first 20 seconds
- Modeling more frequencies allows PDE-Refiner to delay error propagation
- Results in longer stable rollouts

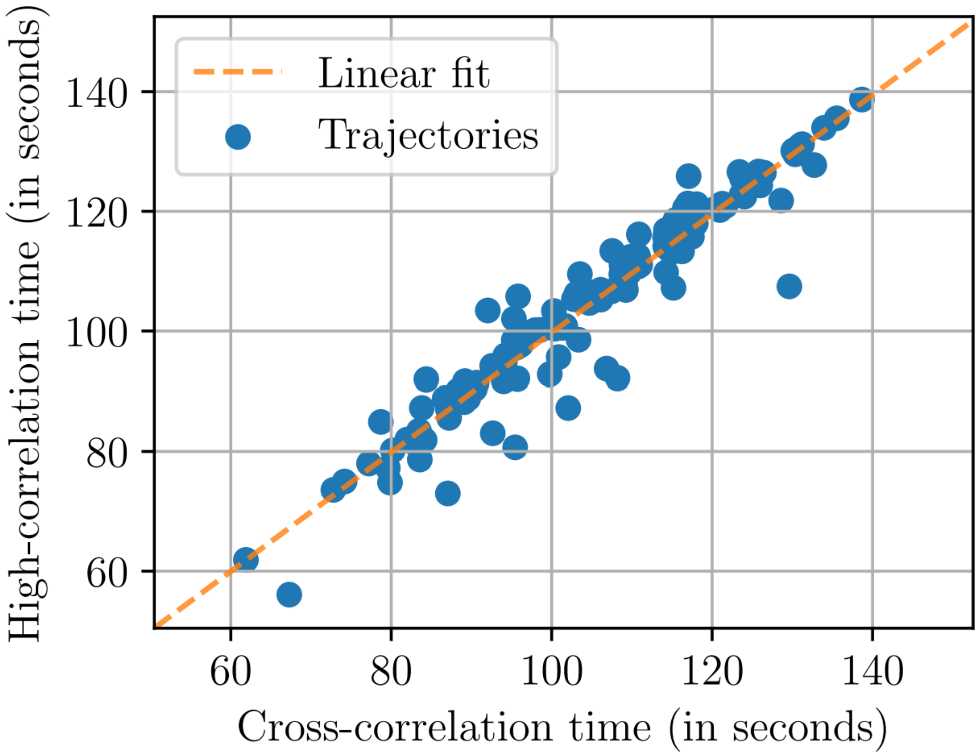


PDE-Refiner – Uncertainty Estimation



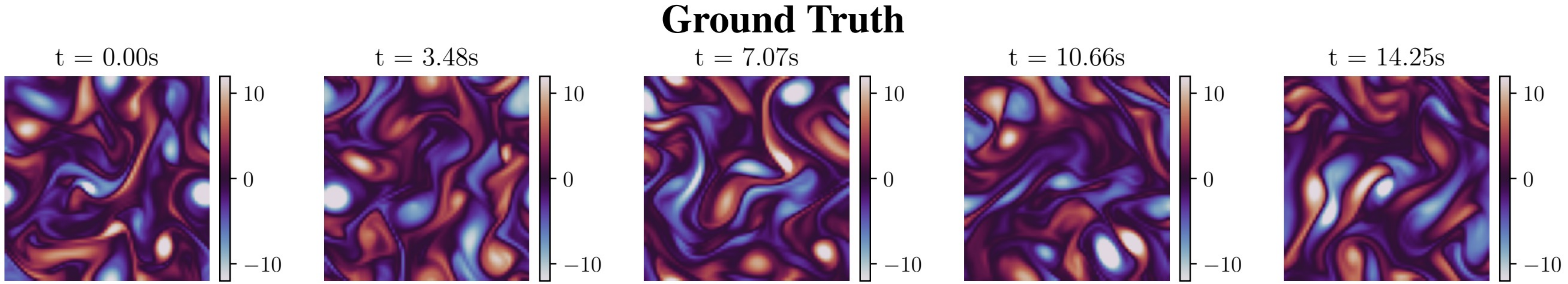
PDE-Refiner – Uncertainty Estimation

- Estimating accurate rollout time by measuring cross-correlation between sampled trajectories
- Accurate uncertainty estimates
- Outperforms other simple uncertainty estimation methods while more efficient than model ensemble



PDE-Refiner – 2D Kolmogorov Flow

- 2D Kolmogorov Flow
  - Generated at 2048x2048 resolution, downscaled to 64x64
  - 2-channel input, evaluated on vorticity



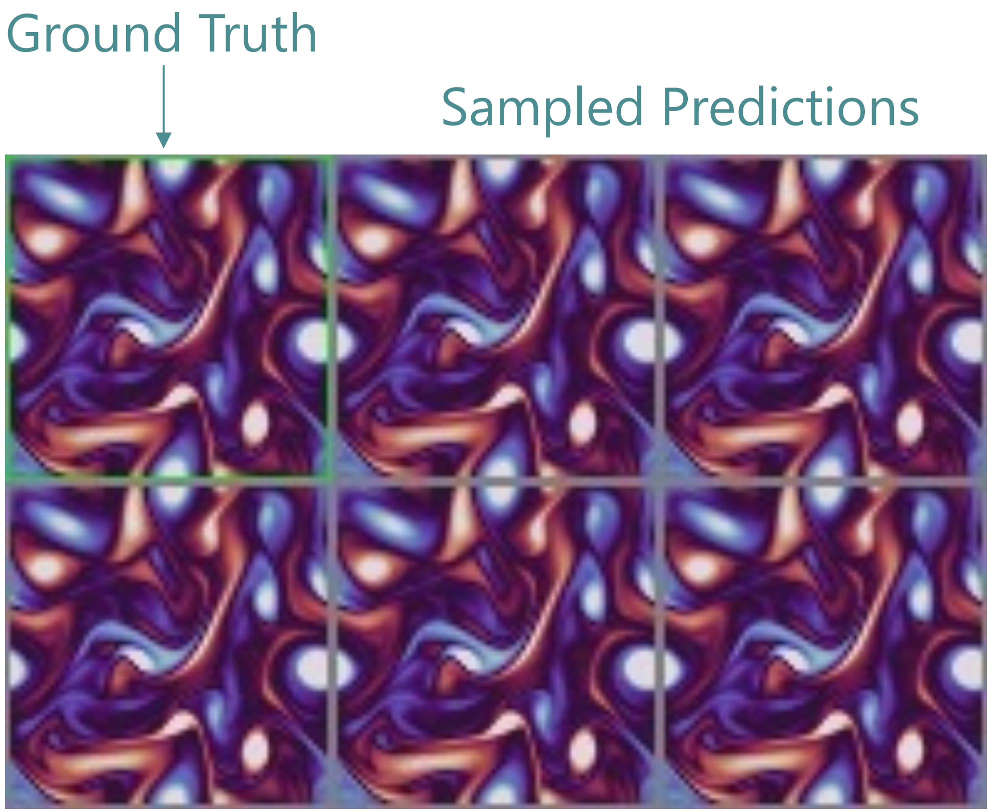
PDE-Refiner – 2D Kolmogorov Flow

- PDE-Refiner outperforms SOTA hybrid solvers
- Gain over MSE baseline smaller than on KS equation
  - Flatter frequency spectrum
  - Smaller resolution
  - Generally higher loss, stronger models possible

Method	Corr. $>$ 0.8 time
Classical PDE Solvers	
DNS - 64×64	2.805
DNS - 128×128	3.983
DNS - 256×256	5.386
DNS - 512×512	6.788
DNS - 1024×1024	8.752
Hybrid Methods	
LC [42, 79] - CNN	6.900
LC [42, 79] - FNO	7.630
LI [42] - CNN	7.910
TSM [75] - FNO	7.798
TSM [75] - CNN	8.359
TSM [75] - HiPPO	9.481
ML Surrogates	
MSE training - FNO	6.451 ± 0.105
MSE training - U-Net	9.663 ± 0.117
PDE-Refiner - U-Net	**10.659** $\pm$ 0.092

PDE-Refiner – 2D Kolmogorov Flow

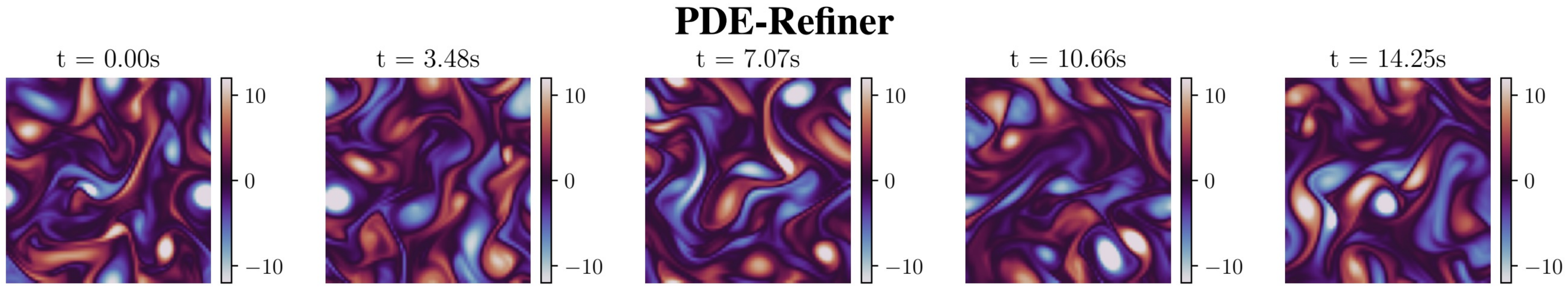
- Sampling multiple trajectories estimates uncertainty
- PDE-Refiner slightly overconfident due to small dataset size



PDE-Refiner – Speed Comparison

- Speed comparison on generating 16 trajectories of 20 seconds
  - MSE model: 4 seconds
  - PDE-Refiner: (K+1) x MSE =16 seconds
  - Hybrid solver: 20 seconds
  - Classical solver: 31 minutes

- PDE-Refiner offers tunable tradeoff between runtime and accuracy



Summary

- Modeling a large spatial frequency band is key for long accurate rollouts
- PDE-Refiner achieves this by an iterative refinement process, gaining up to 30% longer rollouts across different neural operators and PDEs
- Denoising process inherently learns accurate uncertainty estimate
- PDE-Refiner offers flexible tradeoff between accuracy and speed
- Code coming soon in PDEArena

PDE-Refiner: Achieving Accurate Long Rollouts with Neural PDE Solvers

Phillip Lippe, Bas S. Veeling, Paris Perdikaris, Richard E. Turner, Johannes Brandstetter

Project Website

Questions?