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

13 minute read

Published: February 13, 2026

Download this blog as PDF

Shryl Shalom Jennifer

MSc Computer Science

Email

GitHub

CV

Summary. *PDE-Refiner improves long-horizon rollouts of neural PDE solvers by refining each predicted step through an iterative denoising process. The key benefit is better correction of frequency-domain errors that otherwise accumulate and destabilize autoregressive rollouts.*

Overview

Neural surrogate models for time-dependent PDEs can be dramatically faster than classical solvers—especially when high-resolution simulations must be repeated many times (design loops, optimization, uncertainty quantification). The practical bottleneck is not one-step prediction, but **stable long rollouts**: when a model predicts a trajectory step-by-step, small errors compound until the rollout becomes unreliable.

These PDEs arise in real physical systems. For example:

- fluid turbulence in engineering simulations,
- plasma dynamics in physics,
- climate and weather prediction,
- combustion and chemical reaction modeling.

Accurate and fast surrogate solvers can significantly reduce simulation cost in these applications, enabling faster design optimization and uncertainty analysis.

This post summarizes the main ideas and results of *PDE-Refiner: Achieving Accurate Long Rollouts with Neural PDE Solvers* (NeurIPS 2023) [paper].

1. Problem setting: learned rollouts for time-dependent PDEs

We consider PDEs of the form

$$u_t = F(t, x, u, u_x, u_{xx}, \ldots),$$

where $u(t, x)$ is the state evolving over time. A neural PDE solver often learns a one-step evolution map:

$$\hat{u}(t + \Delta t) = \mathcal{M}(\hat{u}(t)).$$

A rollout is autoregressive:

$$\hat{u}(t + 2\Delta t) = \mathcal{M}(\hat{u}(t + \Delta t)), \quad \ldots$$

Even if one-step error is small, **errors accumulate** over steps. In chaotic or turbulent systems, small mismatches can get amplified and lead to rapid divergence from the ground truth.

---

2. Key diagnosis: frequency components are learned unevenly

A central empirical observation is that neural PDE solvers do not learn all spatial frequency components equally well.

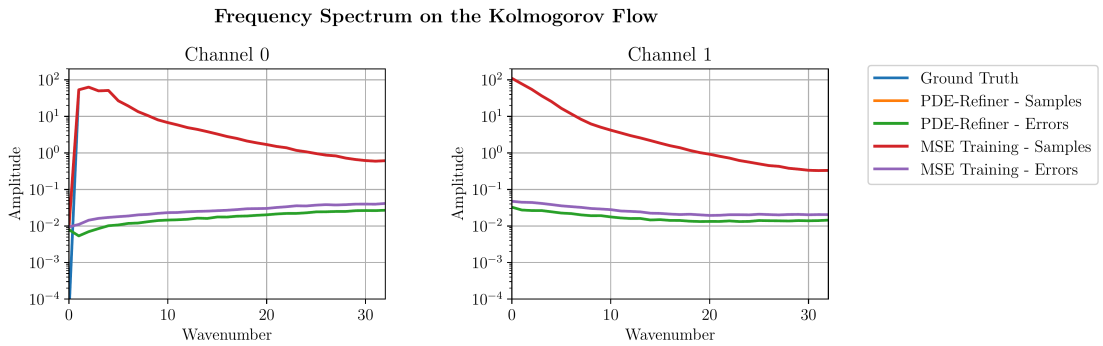
Any spatial field can be decomposed into Fourier modes (frequencies):

- **Low frequencies** capture large-scale structure (smooth patterns).
- **High frequencies** capture fine details (small-scale structure).

Standard training objectives (like MSE) push the model to reduce error where the signal energy is largest. That typically means the model focuses on dominant, high-energy components and can under-correct low-energy bands. This is easy to miss in short-horizon plots because the large-scale dynamics look correct.

However, nonlinear PDEs couple modes. A small error in a “weak” frequency band can seed a growing mismatch that later contaminates larger scales, causing long-horizon instability.

Spectrum intuition



How to read this: the rollout can look plausible at first, while frequency-domain mismatches quietly accumulate until the trajectory collapses.

---

3. Working example: Kuramoto–Sivashinsky (KS) dynamics

The paper uses the **Kuramoto–Sivashinsky (KS)** equation as a primary test case:

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

KS is a strong benchmark because it is chaotic and sensitive: tiny errors grow with time, so long-horizon stability is hard.

A common baseline trains with one-step mean squared error:

$$\mathcal{L}_{\text{MSE}} = \|u(t) - \mathcal{M}(u(t - \Delta t))\|^2.$$

This can produce good one-step predictions, but long rollouts still fail—especially when the model does not reproduce the frequency content accurately.

---

4. Why “fixing rollout strategy” alone is not enough

A natural idea is: “If rollouts fail due to compounding error, let’s change training/rollout strategies.”

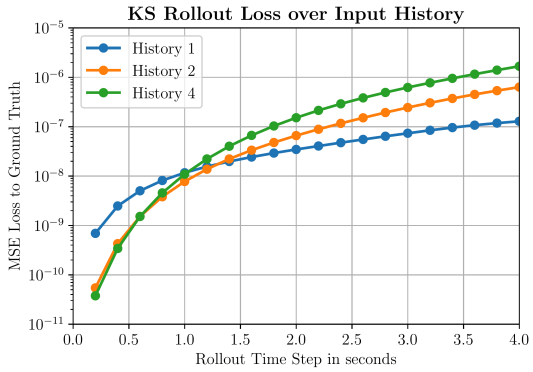
There are several common strategies:

4.1 Train with longer context (history)

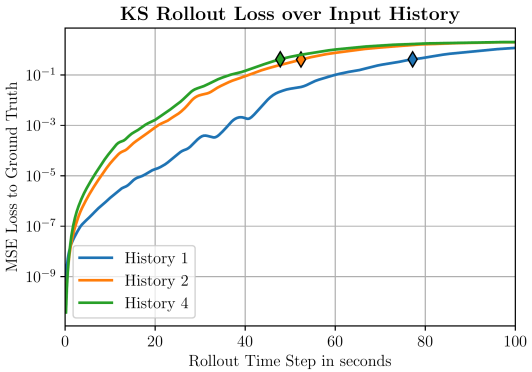
Instead of predicting from a single previous state, models use multiple previous states:

$$\hat{u}(t+\Delta t) = \mathcal{M}(u(t), u(t-\Delta t), \ldots, u(t-H\Delta t)).$$

This can stabilize rollouts slightly, because the model sees more temporal context. But it does not necessarily fix the fundamental issue: if the model still learns an inaccurate spectrum, the error can still accumulate—just slower.

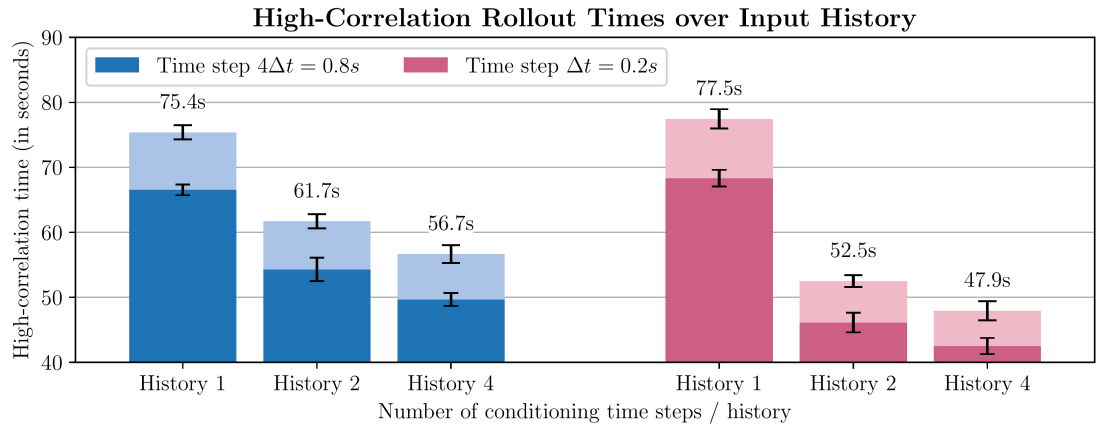


Short rollout horizon



Long rollout horizon

History Bar Plot System:



4.2 Multi-step training / pushforward training

Instead of training only one-step, the model is trained on multiple steps, hoping it learns to correct drift. This often improves stability, but the paper finds it is still limited when frequency-domain mismatches persist.

4.3 Change the loss (e.g., Sobolev / derivative losses)

A Sobolev loss emphasizes derivatives and can indirectly emphasize higher frequencies. These losses can help, but they still optimize a single-shot prediction. They do not explicitly give the model a mechanism to **iteratively correct** a predicted state.

4.4 The core point

Many rollout tricks improve stability *a bit*, but they do not reliably solve:

- **small-but-important frequency mismatches** that accumulate during autoregressive rollout.

So the paper proposes a different idea:

> *Don't only change training schedules—change the prediction process itself so the model can refine and correct a step multiple times.*

---

5. PDE-Refiner: iterative denoising refinement

PDE-Refiner predicts the next state and then **refines it** through a small number of correction steps.

The refinement is set up as a **denoising problem**:

- add noise to the predicted state,
- train the model to estimate/remove that noise,
- repeat with decreasing noise magnitude.

This design matters because noise injects energy across frequencies, forcing corrections across the spectrum rather than only the dominant modes.

5.1 Step-by-step algorithm

At time t we want $u(t + \Delta t)$.

Step 1 — Initial prediction Compute a first estimate:

$$\hat{u}_0(t + \Delta t) = \mathcal{M}_{\text{base}}(u(t), \text{context}).$$

This estimate is usually good in large-scale structure, but may contain systematic errors in fine-scale content.

Step 2 — Add noise For refinement step k:

$$\tilde{u}_k = \hat{u}_k + \sigma_k \epsilon_k, \quad \epsilon_k \sim \mathcal{N}(0, 1).$$

Step 3 — Predict the noise The model predicts the injected noise:

$$\hat{\epsilon}_k = \mathcal{R}(\tilde{u}_k,\ \text{context},\ k).$$

Step 4 — Denoise / refine Update the refined estimate:

$$\hat{u}_{k+1} = \tilde{u}_k - \sigma_k \hat{\epsilon}_k.$$

Step 5 — Repeat with smaller noise Use a decreasing schedule:

$$\sigma_1 > \sigma_2 > \cdots > \sigma_K.$$

This turns refinement into a coarse-to-fine correction process:

- early steps fix large-scale structure,
- later steps recover smaller-scale details.

Algorithm: PDE-Refiner rollout step

Input: state u(t)
Output: refined prediction $\hat{u}(t + \Delta t)$

1. Initial prediction: $\hat{u}_0 = \text{BaseModel}(u(t))$

2. Refinement loop:

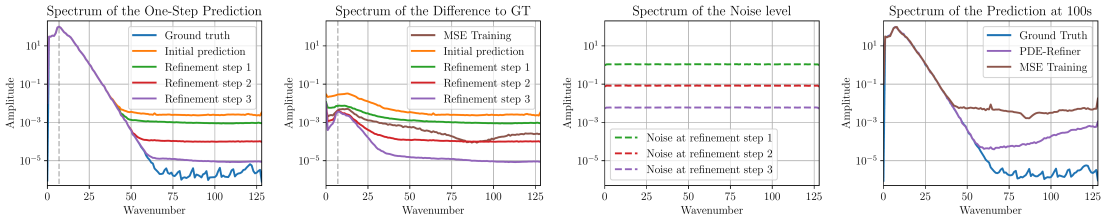
$$\begin{aligned}
&\text{for } k = 1, \ldots, K \text{ do} \\
&\quad \epsilon_k \sim \mathcal{N}(0, 1) \\
&\quad \tilde{u}_k = \hat{u}_k + \sigma_k \epsilon_k \\
&\quad \hat{\epsilon}_k = \text{Refiner}(\tilde{u}_k, \text{context}, k) \\
&\quad \hat{u}_{k+1} = \tilde{u}_k - \sigma_k \hat{\epsilon}_k
\end{aligned}$$

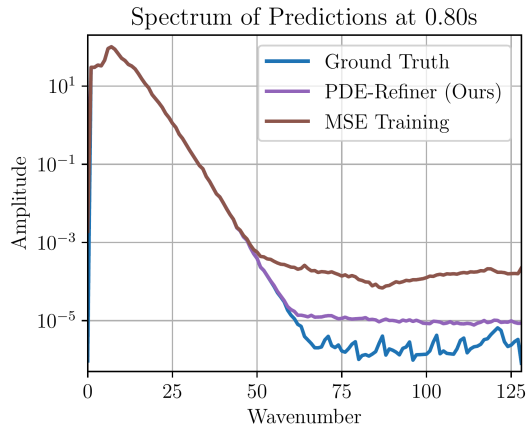
3. Return: $\hat{u}_K$

5.2 Why adding noise helps

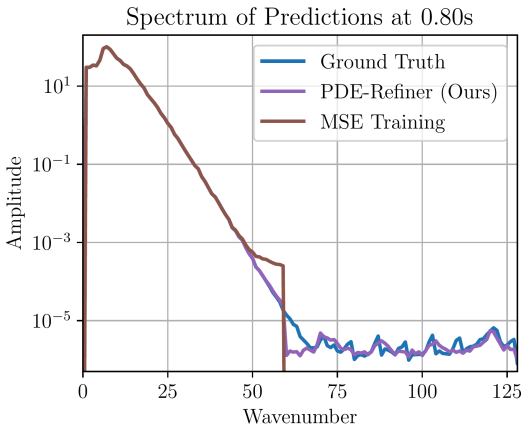
If the model only sees clean predictions, it tends to correct what contributes most to MSE (dominant modes). Adding noise forces the model to learn corrections robustly across amplitudes and frequencies.

5.3 Refinement illustration





Initial prediction spectrum



After refinement correction

6. Training objective (denoising loss)

Instead of predicting a direct residual correction, PDE-Refiner is trained to predict the injected noise. For a sampled refinement step k:

$$\mathcal{L}_k = \mathbb{E}_{\epsilon_k}\Big[\|\epsilon_k - \mathcal{R}(u(t) + \sigma_k \epsilon_k,\ \text{context},\ k)\|^2\Big].$$

This trains the refiner to operate at different noise levels (different scales), which supports coarse-to-fine correction.

7. Why it works: improved accuracy across frequency bands

The paper's key mechanism is that refinement reduces error **more uniformly across frequencies**.

7.1 What baseline training tends to do

With standard MSE training, the model reduces error where the energy is highest. That often means:

- dominant low-frequency modes are good,
- higher-frequency modes are under-corrected,
- long rollouts drift when those small errors get amplified.

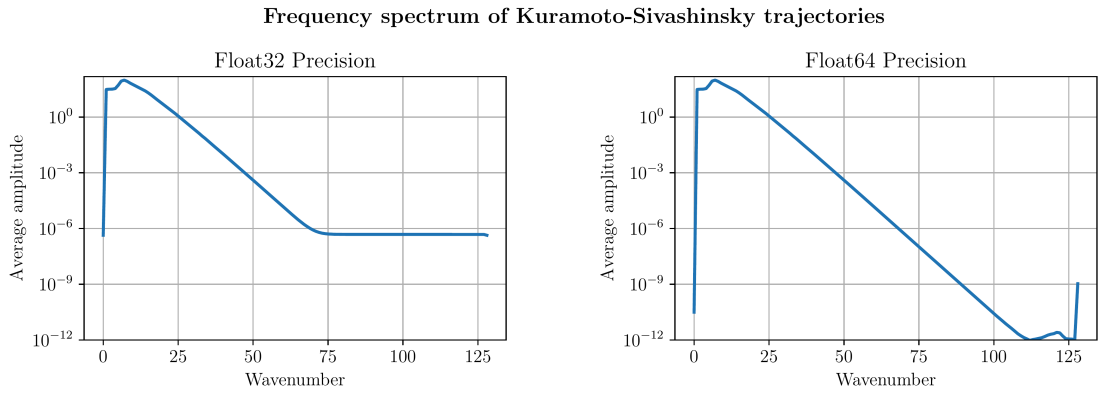
7.2 What PDE-Refiner changes

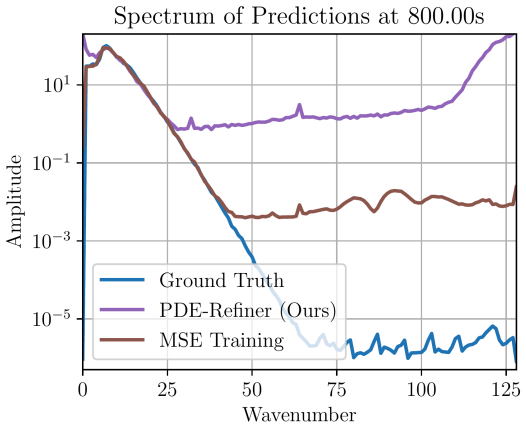
PDE-Refiner introduces multiple correction opportunities per step. Because refinement is framed as denoising with varying noise magnitudes, the model learns to correct:

- large-scale structure (early steps),
- finer-scale details (later steps).

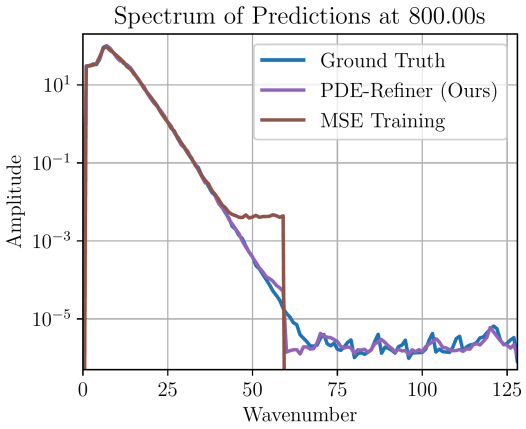
This matters because in chaotic systems, a small fine-scale mismatch can act like a perturbation that grows over time.

7.3 Frequency-domain evidence

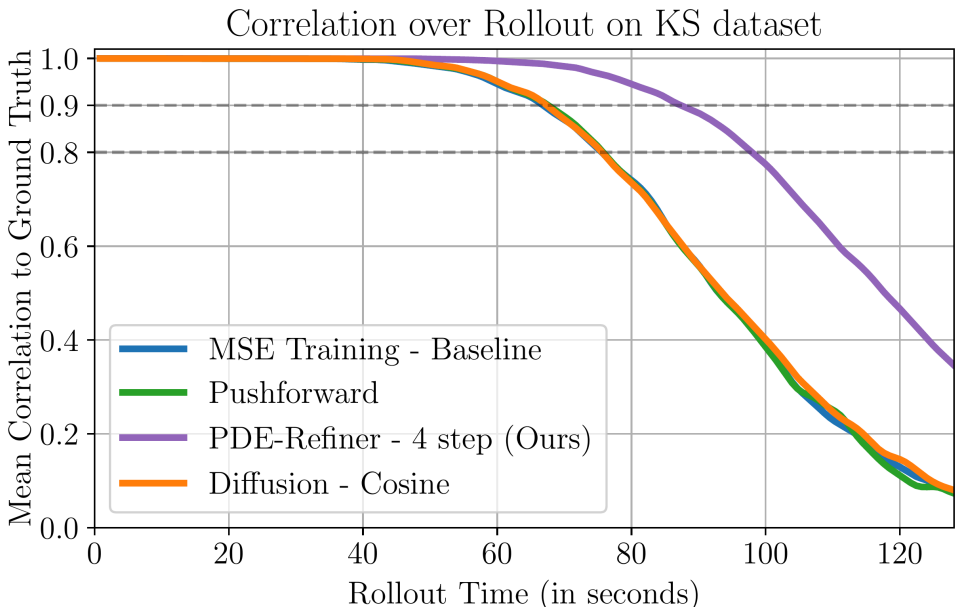




Baseline spectrum after long rollout



PDE-Refiner spectrum after long rollout



8. Experiments

8.1 Experiment 1 — 1D Kuramoto–Sivashinsky rollouts

KS is chaotic, so the correct question is not “does it look okay at one step,” but:

> ***How long can the model roll out before it loses correlation with the ground truth?***

Setup (high-level)

The model is trained on KS trajectories and evaluated by rolling out many steps autoregressively. Evaluation focuses on long horizons where compounding error matters.

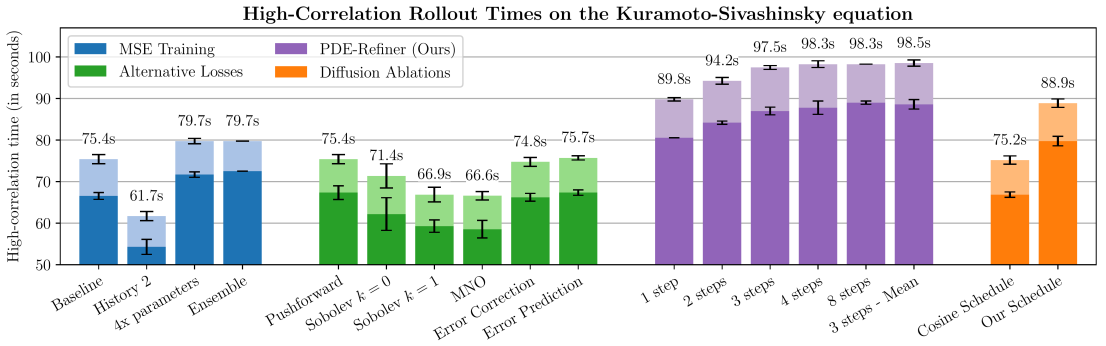
A common metric in the paper is correlation over time (or correlation time): how long predictions remain aligned with ground truth before dropping below a threshold.

Baselines vs PDE-Refiner

Baselines can show:

- good one-step error,
- plausible short-term patterns,
- but loss of correlation at longer horizons.

PDE-Refiner improves the long-horizon behavior by refining each step, reducing the drift that accumulates from subtle frequency errors.

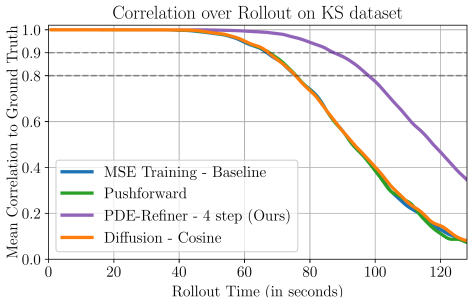


Interpreting the result

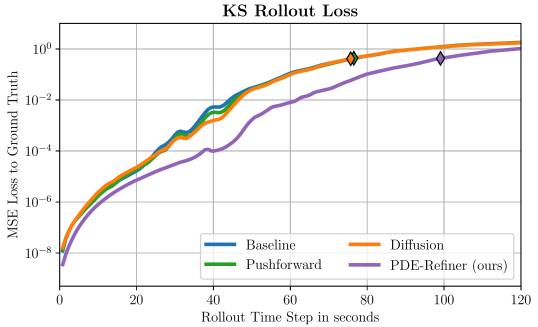
This bar plot shows that PDE-Refiner maintains accurate rollouts significantly longer than:

- plain MSE training,
- and multiple alternative rollout strategies.

The important point is that refinement improves long-term stability without requiring a separate simulator or handcrafted correction.



Correlation between prediction and ground truth over rollout horizon



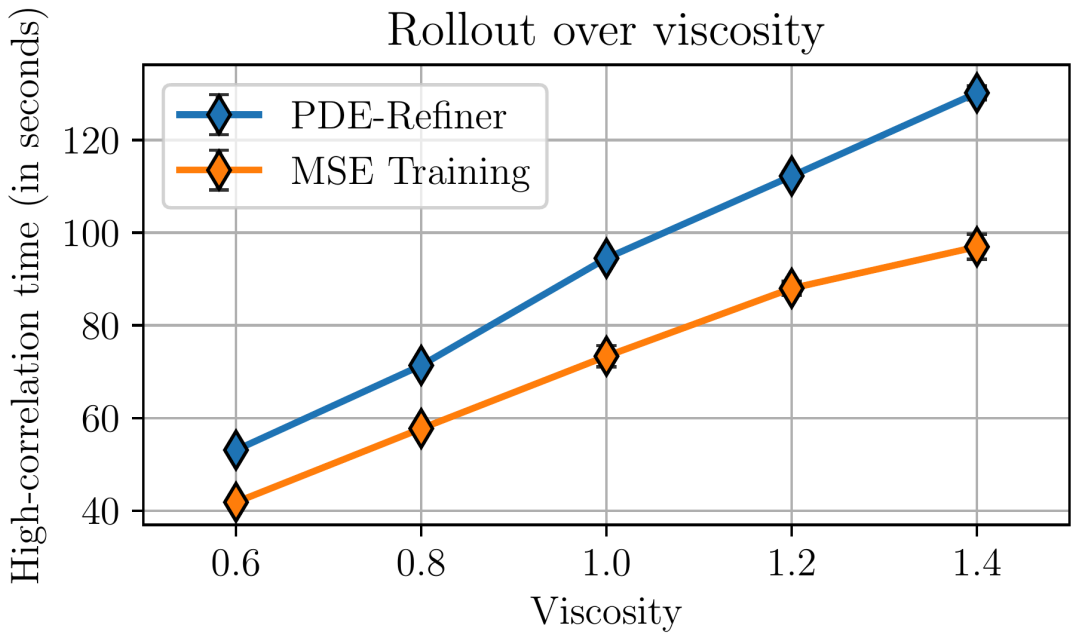
Error accumulation across rollout steps for different methods

These show:

- qualitative rollouts,
- how methods compare over time,
- and what “failure” looks like.

8.2 Experiment 2 — Generalization across viscosity

The paper also varies viscosity ν, which changes how strongly high frequencies are damped. This tests whether the method generalizes across regimes.



Interpretation: PDE-Refiner improves rollout behavior across viscosities, suggesting robustness rather than tuning to a single setting.

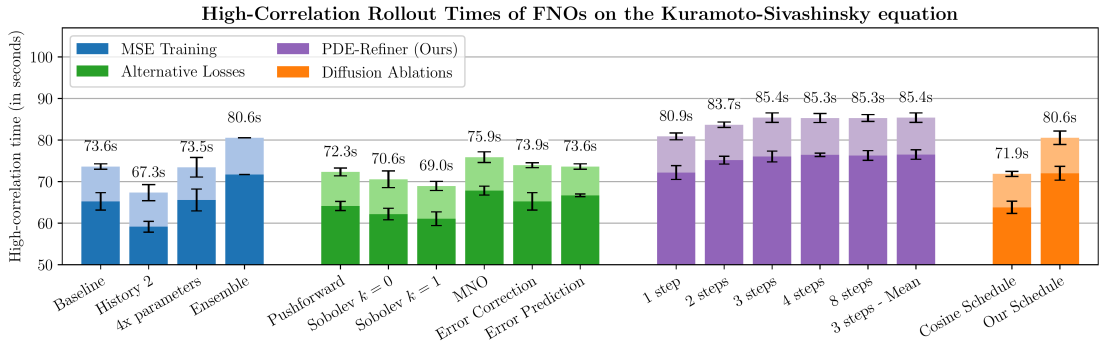
8.3 Experiment 3 — 2D Kolmogorov flow (turbulence benchmark)

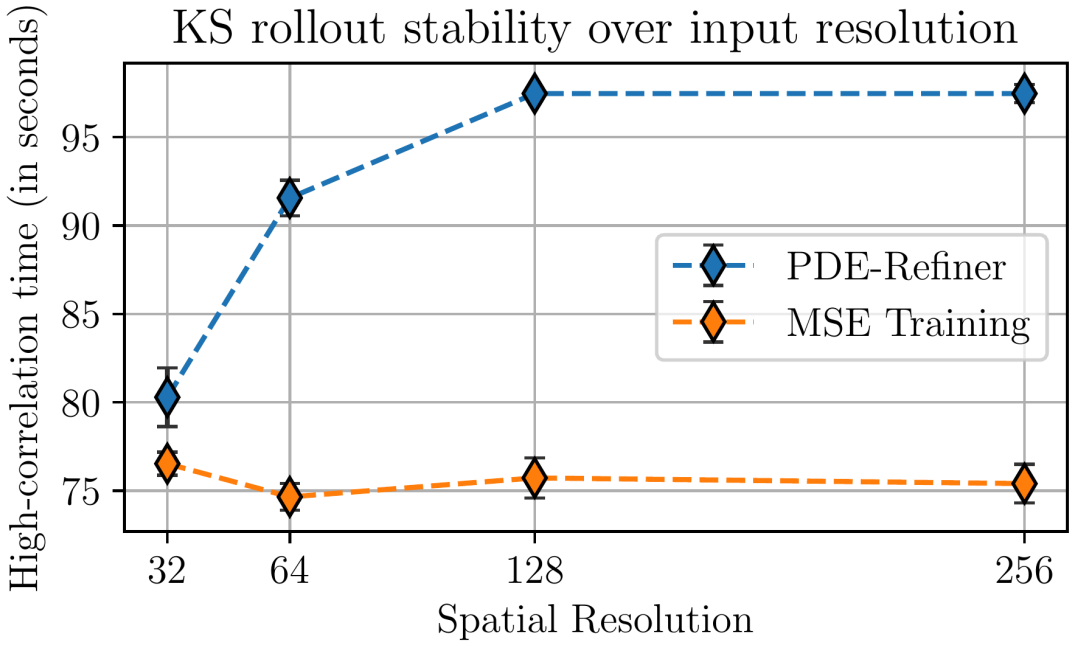
Kolmogorov flow is a harder, more realistic setting because turbulence is strongly nonlinear and sensitive to fine-scale error.

A simplified form of the dynamics:

$$\partial_t u + \nabla \cdot (u \otimes u) = \nu \nabla^2 u - \nabla p + f.$$

The takeaway is consistent: when small-scale errors are not controlled, long rollouts drift. Refinement improves stability by correcting a broader range of frequency content.



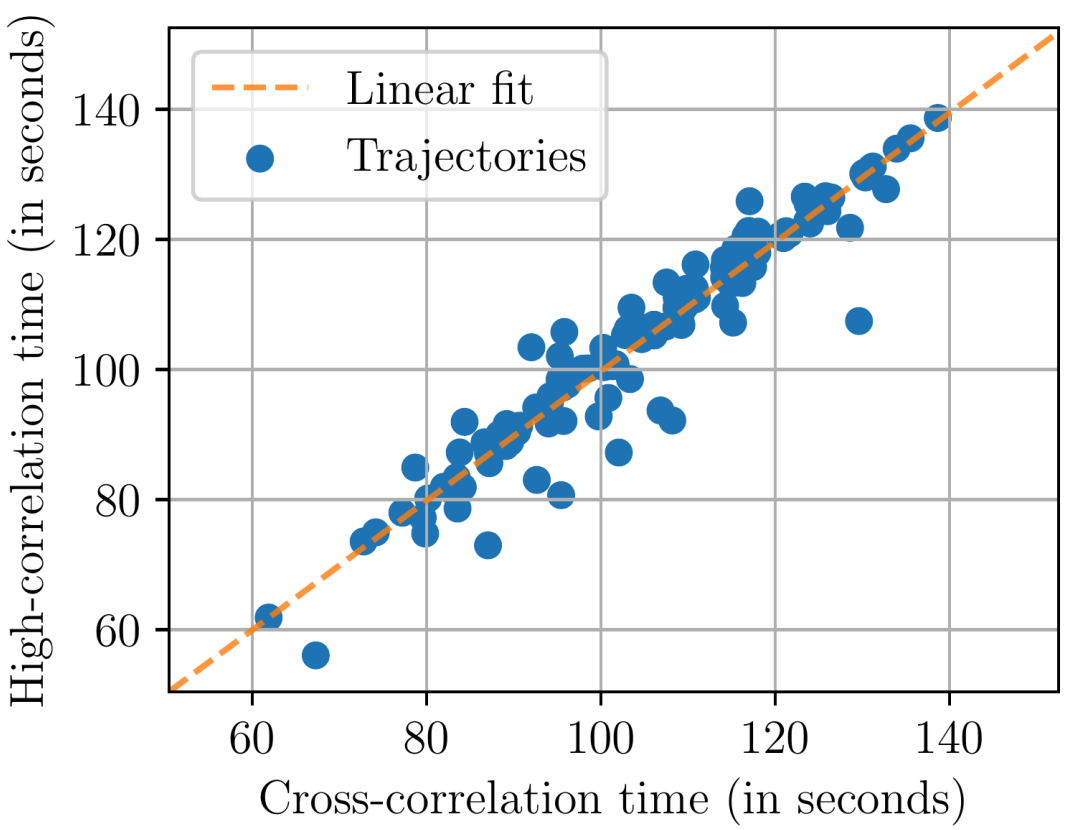


9. Uncertainty estimation via sampling

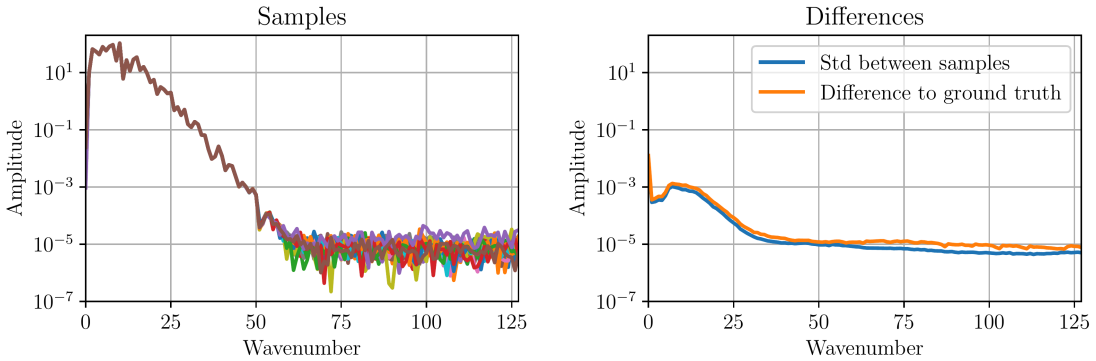
PDE-Refiner enables a practical uncertainty signal because refinement involves stochastic noise. We can run multiple refinement samples (different noise draws) and compare their rollouts.

- If multiple sampled rollouts stay similar for a long time, confidence is higher.
- If they diverge quickly, uncertainty is higher—and the true rollout typically fails sooner.

This is useful operationally: it provides a “trust horizon” estimate without training a full ensemble.



Uncertainty estimation scatter



Diffusion samples standard deviation

10. Pros, cons, and trade-offs

Pros

- Improves long-horizon rollout stability in chaotic regimes.
- Corrects errors across a broader frequency range.
- Robust across viscosity (regime shifts).
- Provides a usable uncertainty signal from sampling.

Cons / trade-offs

- More compute: refinement requires multiple model evaluations per timestep.
- Requires choosing refinement steps and noise schedule (design choices).

Future work

Possible directions include:

- faster refinement schedules (fewer steps with similar performance),
- distilling refinement into a cheaper predictor,
- extending to larger-scale 2D/3D PDE systems,
- combining refinement with physics-informed constraints or conservation priors,
- automatic stopping criteria based on uncertainty signals,
- exploring refinement with other neural operator families.

Conclusion

Long-horizon rollout failure is strongly connected to frequency-domain error accumulation: standard training often under-corrects low-energy frequency components that later become dynamically important.

PDE-Refiner improves stability by refining each predicted step through iterative denoising, producing better frequency coverage, longer stable rollouts, and a natural uncertainty signal via sampling.

References

Primary paper

[1] Lippe, P., et al. (2023).
PDE-Refiner: Achieving Accurate Long Rollouts with Neural PDE Solvers.
NeurIPS 2023.

- arXiv page
- PDF

Related work on neural PDE solvers

[2] Brandstetter, J., Worrall, D., and Welling, M. (2022).
Message Passing Neural PDE Solvers.
International Conference on Learning Representations (ICLR).

[3] Brandstetter, J., Berg, R. van den, Welling, M., and Gupta, J. (2023).
Clifford Neural Layers for PDE Modeling.
International Conference on Learning Representations (ICLR).

[4] Bar-Sinai, Y., Hoyer, S., Hickey, J., and Brenner, M. (2019).
Learning data-driven discretizations for partial differential equations.
Proceedings of the National Academy of Sciences.

Related work on diffusion and denoising

[5] Dhariwal, P. and Nichol, A. (2021).
Diffusion models beat GANs on image synthesis.
NeurIPS.

[6] Berthelot, D., et al. (2023).
TRACT: Denoising Diffusion Models with Transitive Closure Time-Distillation.
arXiv:2303.04248.

Related work on stability and surrogate modeling

[7] Chattopadhyay, A. and Hassanzadeh, P. (2023).
Long-term instabilities of deep learning-based climate digital twins.
arXiv:2304.07029.

[8] Arcumano, T., et al. (2022).
A Hybrid Approach to Atmospheric Modeling Combining Machine Learning and Physics-Based Simulation.
Journal of Advances in Modeling Earth Systems.

Tools used

[9] Bradbury, J., et al. (2018).
JAX: composable transformations of Python+NumPy programs.
https://github.com/google/jax