# ANALYSIS OF NEURAL ODE PERFORMANCE IN LONG-TERM PDE SEQUENCE MODELING

**Fang Sun***, **Maxwell Dalton***, **Yizhou Sun**
Department of Computer Science, UCLA
{fts,maxdalton,yzsun}@cs.ucla.edu

## ABSTRACT

Predicting solutions to partial differential equations (PDEs) from limited snapshots is central to many scientific and engineering disciplines. However, standard autoregressive neural models often suffer from compounding errors when rolled out over large time scales. In this paper, we investigate Neural Ordinary Differential Equations (Neural ODEs) as an alternative to purely discrete, stepwise approaches for PDE sequence modeling. We propose a continuous-time neural solver architecture that combines a graph encoder to process local spatial features, a learned ODE network to integrate node embeddings forward in time, and a decoder that produces PDE field predictions at future timesteps. We compare this Neural ODE solver to a baseline autoregressive GNN on a long-horizon sequence-to-sequence prediction task for Burgers' equation without diffusion. Empirical results indicate that our Neural ODE approach significantly reduces error growth over extended time scale and achieves improved stability. These findings highlight the promise of continuous-time neural modeling for robust PDE simulation and pave the way for applying learned surrogates to complex scientific systems. Our implementation is available at https://github.com/FrancoTSolis/neural-ode-pde.

## 1 INTRODUCTION

Many physical systems of interest are governed by partial differential equations (PDEs), and traditional numerical solvers have long been the backbone of computational science (LeVeque, 2002; Hughes, 2003). Despite their high accuracy, classical methods such as finite differences, finite volumes, and finite elements can be computationally expensive and are often tailored to specific problems, especially when high resolution or long-term integration is required (Li et al., 2020; Lynch, 2008). Recent advances in machine learning have led to the development of surrogate models that aim to approximate PDE solutions more rapidly (Lu et al., 2019; Raissi et al., 2019; Belbute-Peres et al., 2020). In particular, Graph Neural Networks (GNNs) (Hamilton et al., 2017; Kipf & Welling, 2016) have emerged as a promising framework for modeling PDE dynamics by discretizing the domain into graph nodes and learning local update rules (Sanchez-Gonzalez et al., 2020). However, these autoregressive models predict one step at a time and recursively feed their own predictions as input, which often leads to error accumulation over long sequences (Bengio et al., 2015).

A critical limitation of existing autoregressive GNN-based solvers is their inherent vulnerability to compounding errors, which makes them less reliable for long-horizon predictions. As illustrated in Figure 1, autoregressive methods propagate errors forward due to their discrete-step nature, whereas Neural Ordinary Differential Equations (Neural ODEs) offer an attractive alternative by modeling the evolution of hidden states continuously over time. This continuous formulation allows for adaptive integration, which can mitigate the error accumulation observed in discrete-step methods (Chen et al., 2018; Rubanova et al., 2019). In this work, we propose a Neural ODE PDE Solver that first encodes local PDE states into node embeddings via a GNN and then evolves these embeddings in continuous time using a learned ODE integrator. By training directly on long-horizon sequences, our approach addresses the limitations of conventional autoregressive methods and achieves superior accuracy and stability on challenging benchmarks such as Burgers' equation without diffusion.

*First author, equal contribution.

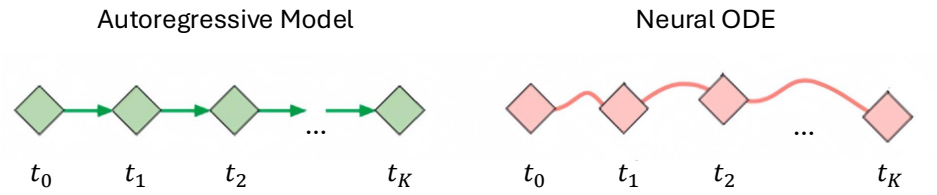


Figure 1: Comparison of autoregressive models (left) and Neural ODEs (right) for modeling temporal evolution. Autoregressive models predict each step sequentially, leading to error accumulation, whereas Neural ODEs model the dynamics continuously, mitigating compounding errors.

## 2 Related work

In this section, we review related work from three main perspectives: traditional numerical methods, GNN-based PDE solvers, and continuous-time deep models based on Neural ODEs. While each approach has significantly advanced the field of PDE simulation, limitations remain in terms of computational efficiency, scalability, and long-horizon prediction stability.

### 2.1 Traditional Numerical Methods

Traditional numerical methods for solving PDEs, including finite difference, finite volume, and finite element techniques, have been extensively developed over the past decades (LeVeque, 2002; Hughes, 2003). These methods rely on discretizing the spatial and temporal domains and applying well-defined stencils or basis functions to approximate derivatives. While they are capable of capturing sharp features such as shock waves using advanced schemes (e.g., Godunov, WENO) (LeVeque, 2002; Igor & Dmitry, 2021), their application is often limited by high computational costs, especially for high-resolution simulations or long-term integrations (Lynch, 2008). Moreover, each simulation must be run from scratch, which limits their scalability in scenarios that require repeated evaluations over varying parameters.

### 2.2 GNN-Based PDE Solvers

GNN-based PDE solvers represent an emerging class of methods that leverage the power of graph neural networks to model the dynamics of PDEs on discretized domains (Sanchez-Gonzalez et al., 2020; Belbute-Peres et al., 2020). These approaches convert the spatial domain into a graph where each node corresponds to a grid cell or mesh element, and they employ message passing to learn local differential operators. Although such models have demonstrated promising results in capturing complex dynamics in systems ranging from fluid flows to structural mechanics, they typically operate in an autoregressive manner. This sequential prediction strategy introduces an inherent risk of error accumulation over long rollouts, which can degrade performance and reduce long-term stability (Bengio et al., 2015; Brandstetter et al., 2022).

### 2.3 Neural ODEs and Continuous-Time Deep Models

Neural Ordinary Differential Equations (Neural ODEs) provide a continuous-time framework in which the evolution of hidden states is governed by an ODE, $\frac{d\mathbf{z}(t)}{dt} = f(\mathbf{z}(t), t; \theta)$, that is integrated using standard solvers such as Runge-Kutta or Dormand-Prince (Chen et al., 2018; Rubanova et al., 2019; Grathwohl et al., 2018). This formulation treats the depth of a neural network as a continuous variable, allowing for adaptive step sizes and improved numerical stability. While Neural ODEs have shown success in areas such as image classification, time-series modeling, and generative modeling, their application to PDE simulation remains less explored. The continuous-time integration inherent in Neural ODEs offers a promising solution to the compounding error problem in discrete, autoregressive methods, yet challenges remain in scaling these models to high-dimensional, complex physical systems (Raissi et al., 2019).

## 3 PROBLEM DEFINITION

Consider a 1D PDE system with states $\mathbf{u}(t) \in \mathbb{R}^m$, discretized on $N$ spatial cells. We have training examples:

$$\left\{ \left( \mathbf{u}_n^{(t_1:t_L)}, \mathbf{u}_n^{(t_{L+1}:t_{L+K})} \right) \right\}_{n=1}^{N_{\text{train}}}, \tag{1}$$

where each sample is a trajectory from time $t_1$ to $t_{L+K}$. Our goal is to learn a predictor $\mathcal{M}$ that maps the first $L$ snapshots to the next $K$ snapshots:

$$\hat{\mathbf{u}}^{(t_{L+1}:t_{L+K})} = \mathcal{M}\left( \mathbf{u}^{(t_1:t_L)} \right). \tag{2}$$

## 4 METHODOLOGY

Our proposed Neural ODE PDE Solver is composed of three main components: a graph-based encoder inspired by prior work on MP-PDE solvers (Brandstetter et al., 2022), a continuous-time evolution module based on Neural ODEs, and a decoder.

### 4.1 GRAPH REPRESENTATION AND ENCODING

We begin by discretizing the spatial domain into a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where each node $i \in \mathcal{V}$ corresponds to a spatial cell and edges in $\mathcal{E}$ connect neighboring nodes (Sanchez-Gonzalez et al., 2020; Belbute-Peres et al., 2020). For each node $i$, we collect a time series of the last $L$ PDE states, denoted by $\mathbf{u}_i^{(t_1:t_L)}$. A small multilayer perceptron (MLP) serves as an encoder to map these input features into an initial embedding:

$$\mathbf{h}_i^{(0)} = \text{Enc}\left( \mathbf{u}_i^{(t_1:t_L)} \right). \tag{3}$$

### 4.2 GRAPH NEURAL NETWORK PROCESSOR

Next, the encoded node features are refined through $M$ rounds of message passing. At each round $\ell = 0, \ldots, M-1$, messages are computed for every edge $(i, j)$ as follows:

$$\mathbf{m}_{ij}^{(\ell)} = \phi\left( \mathbf{h}_i^{(\ell)}, \mathbf{h}_j^{(\ell)}, \mathbf{x}_i - \mathbf{x}_j \right), \tag{4}$$

where $\mathbf{x}_i - \mathbf{x}_j$ represents the relative spatial position between nodes $i$ and $j$. Each node's embedding is then updated via an aggregation function:

$$\mathbf{h}_i^{(\ell+1)} = \psi\left( \mathbf{h}_i^{(\ell)}, \sum_{j \in N(i)} \mathbf{m}_{ij}^{(\ell)} \right). \tag{5}$$

After $M$ rounds, the final embeddings $\{\mathbf{h}_i^{(M)}\}$ are collected into the latent representation $\mathbf{z}(t_L)$, where $t_L$ marks the end of the observed time window.

### 4.3 NEURAL ODE INTEGRATION

The evolution of the latent representation is modeled continuously in time. Specifically, we assume that $\mathbf{z}(t)$ obeys an ordinary differential equation of the form:

$$\frac{d\mathbf{z}(t)}{dt} = f\left( \mathbf{z}(t), t; \theta_{\text{ODE}} \right), \tag{6}$$

where $f$ is a learned MLP parameterized by $\theta_{\text{ODE}}$. We integrate this ODE from time $t_L$ to $t_L + K$ using a numerical solver (Chen et al., 2018; Grathwohl et al., 2018):

$$\mathbf{z}\left( t_L + K \right) = \text{odeint}\left( \mathbf{z}\left( t_L \right), f, [t_L, t_L + K] \right). \tag{7}$$

Intermediate latent states can be queried at desired time points if needed.

### 4.4 DECODING

Finally, for each node $i$, the latent state $\mathbf{z}_i\left(t_L+k\right)$ is mapped back to the physical PDE state using a decoding MLP:

$$\hat{\mathbf{u}}_i^{(t_L+k)} = \mathrm{Dec}\left(\mathbf{z}_i\left(t_L+k\right)\right), \quad k=1,\ldots,K. \tag{8}$$

This produces the predicted sequence $\hat{\mathbf{u}}^{(t_{L+1}:t_{L+K})}$, which approximates the true PDE evolution.

## 5 EXPERIMENTS

In this section, we address two main research questions:

- **RQ1 (Sequence Recovery Accuracy):** Can the Neural ODE approach more accurately recover the PDE sequence compared to an autoregressive baseline?
- **RQ2 (Long-term Prediction Stability):** Does the Neural ODE method improve long-term prediction stability, particularly over extended rollouts, compared to the autoregressive approach?

### 5.1 DATASET AND BASELINE

Our experiments are conducted on data generated from Burgers' equation without diffusion, a classic hyperbolic PDE known for its shock formation behavior (LeVeque, 2002). The dataset consists of 2304 trajectories, each containing 250 timesteps with a spatial resolution of $n_x = 100$. The baseline model is an autoregressive GNN-based PDE solver (e.g., MP-PDE Solvers (Brandstetter et al., 2022)), which predicts one time window at a time and recursively feeds its output as input. Detailed training setups, hyperparameters, and data splits are provided in the Appendix.

### 5.2 RQ1: SEQUENCE RECOVERY ACCURACY

To evaluate sequence recovery accuracy, we compare the mean squared error (MSE) between the predicted PDE states and the ground truth. Table 1 shows that our Neural ODE model achieves a final MSE of 0.5267, substantially lower than the 1.2176 achieved by the autoregressive baseline. In addition, visual inspection of the predicted sequences (see Figure 3 in the Appendix) confirms that the Neural ODE predictions more closely resemble the ground truth, especially in regions with sharp transitions. These results suggest that continuous-time integration via Neural ODEs enhances the fidelity of sequence recovery.

Table 1: MSE comparison between the autoregressive baseline and the Neural ODE model.

| **Method** | $\frac{1}{n_x}\sum_{x,t}\mathrm{MSE}\downarrow$ |
|---|---|
| MP-PDE | 1.2176 |
| Ours | **0.5267** |

### 5.3 RQ2: LONG-TERM PREDICTION STABILITY

We further assess long-term prediction stability by examining the evolution of the per-timestep $\sum_x$ MSE over the prediction horizon. As shown in Figure 2, the autoregressive baseline initially achieves slightly lower MSE in the early timesteps; however, its error increases sharply as the rollout proceeds. In contrast, the Neural ODE model demonstrates a gradual improvement, ultimately outperforming the baseline in later timesteps. This behavior indicates that continuous-time integration can effectively mitigate the accumulation of errors over long horizons, yielding superior stability for extended predictions.

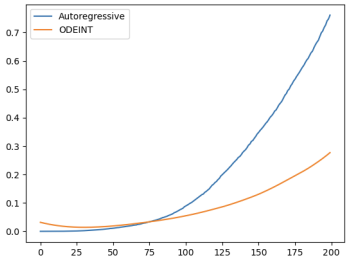


Figure 2: Comparison of average $\sum_x$ MSE loss per timestep.

## 6 CONCLUSION

Our work demonstrates that continuous-time neural modeling via Neural ODEs significantly enhances the robustness and accuracy of PDE simulations. These findings underscore the potential of learned surrogate models to effectively tackle complex scientific systems, offering a promising alternative to conventional discrete solvers.

## 7 ACKNOWLEDGMENTS

This work was partially supported by NSF 2211557, NSF 2119643, NSF 2303037, NSF 2312501, SRC JUMP 2.0 Center, Amazon Research Awards, and Snapchat Gifts.

## REFERENCES

Filipe De Avila Belbute-Peres, Thomas Economon, and Zico Kolter. Combining differentiable pde solvers and graph neural networks for fluid flow prediction. In *international conference on machine learning*, pp. 2402–2411. PMLR, 2020.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. *Advances in neural information processing systems*, 28, 2015.

Johannes Brandstetter, Daniel Worrall, and Max Welling. Message passing neural pde solvers. *arXiv preprint arXiv:2202.03376*, 2022.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 2018.

Will Grathwohl, Ricky TQ Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Thomas JR Hughes. *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation, 2003.

Kulikov Igor and Karavaev Dmitry. The weno reconstruction in the godunov method for modeling hydrodynamic flows with shock waves. In *Journal of Physics: Conference Series*, volume 2028, pp. 012023. IOP Publishing, 2021.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Randall J LeVeque. *Finite volume methods for hyperbolic problems*, volume 31. Cambridge university press, 2002.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

Lu Lu, Pengzhan Jin, and George Em Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

Peter Lynch. The origins of computer weather prediction and climate modeling. *Journal of computational physics*, 227(7):3431–3444, 2008.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

Yulia Rubanova, Ricky TQ Chen, and David K Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. *Advances in neural information processing systems*, 32, 2019.

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International conference on machine learning*, pp. 8459–8468. PMLR, 2020.

# A Detailed Training Setup and Additional Implementation Details

## A.1 Training Setup and Data Splits

Our dataset consists of 2,304 trajectories generated from Burgers' equation without diffusion (LeVeque, 2002). Each trajectory spans 250 timesteps and uses a spatial resolution of 100, matching the default parameters outlined in the code (see `train.py`). We partition the data into training (2,048 trajectories), validation (128 trajectories), and test sets (128 trajectories). To ensure consistency across experiments, all trajectories are downsampled when necessary so that the input dimensionality remains aligned with the models in use.

## A.2 Hyperparameters

We employ the following hyperparameters in our experiments (with default values shown in `train.py`):

- **Input Time Window:** 25 timesteps.
- **Prediction Horizon (Neural ODE Model):** 200 timesteps.
- **Graph Neural Network:** The encoder and processor each contain 6 layers, with 2 rounds of message passing per layer.
- **Node Embedding Dimension:** Configurable parameter in the GNN, set to a moderate size to balance expressivity and computational cost.
- **Neural ODE Derivative Network:** Implemented to model continuous time evolution of the latent embeddings.
- **ODE Solver:** Specified via the `torchdiffeq` library for continuous-time integration.
- **Optimizer:** AdamW with a learning rate of $1 \times 10^{-4}$ and weight decay of $1 \times 10^{-8}$.
- **Batch Size:** Typically set to 16, although this may be adjusted based on memory constraints.
- **Number of Training Epochs:** Up to 20 epochs, subject to early stopping criteria.

## A.3 Learning Objective and Training Loss

Both the autoregressive baseline and the Neural ODE model are trained to minimize mean squared error on the predicted PDE states over the number of time steps we wish to forecast:

$$\mathcal{L} = \sum_{k=1}^{K} \left\| \hat{\mathbf{u}}^{(t_L+k)} - \mathbf{u}^{(t_L+k)} \right\|^2, \tag{9}$$

where $\hat{\mathbf{u}}^{(t_L+k)}$ is the predicted state at timestep $t_L + k$, and $\mathbf{u}^{(t_L+k)}$ denotes the corresponding ground truth.

## A.4 Baseline: Autoregressive Time Binding

Our baseline model applies a GNN-based PDE solver (Brandstetter et al., 2022; Sanchez-Gonzalez et al., 2020) autoregressively over 25-timestep windows. During inference, the model's output for one window becomes the input for the subsequent window. While this iterative strategy is straightforward, it can suffer from compounding errors as the model repeatedly uses its own predictions as inputs to later predictions.

## A.5 Direct Inference with Neural ODE Integration

In contrast, our Neural ODE model integrates the learned dynamics over the entire future time horizon in one continuous pass. After encoding the input state into latent representations, the ODE solver propagates these representations forward through time, producing predictions for all intermediate timesteps at once. This avoids the recurrent re-feeding of predictions into the model and generally provides more stable long-term forecasts.

### A.6 Additional Implementation Details

All experiments were conducted in PyTorch, and the solver for the Neural ODE model was implemented using `torchdiffeq`. For stable optimization, gradient clipping (with a threshold of 1.0) was applied, and standard initialization schemes were used for model parameters. Further information, such as the scheduling of learning rates, logging of metrics, and handling of edge cases (e.g., parameter ablations), can be found in the training script (`train.py`) and documentation within the code repository.

## B More Experiment Results

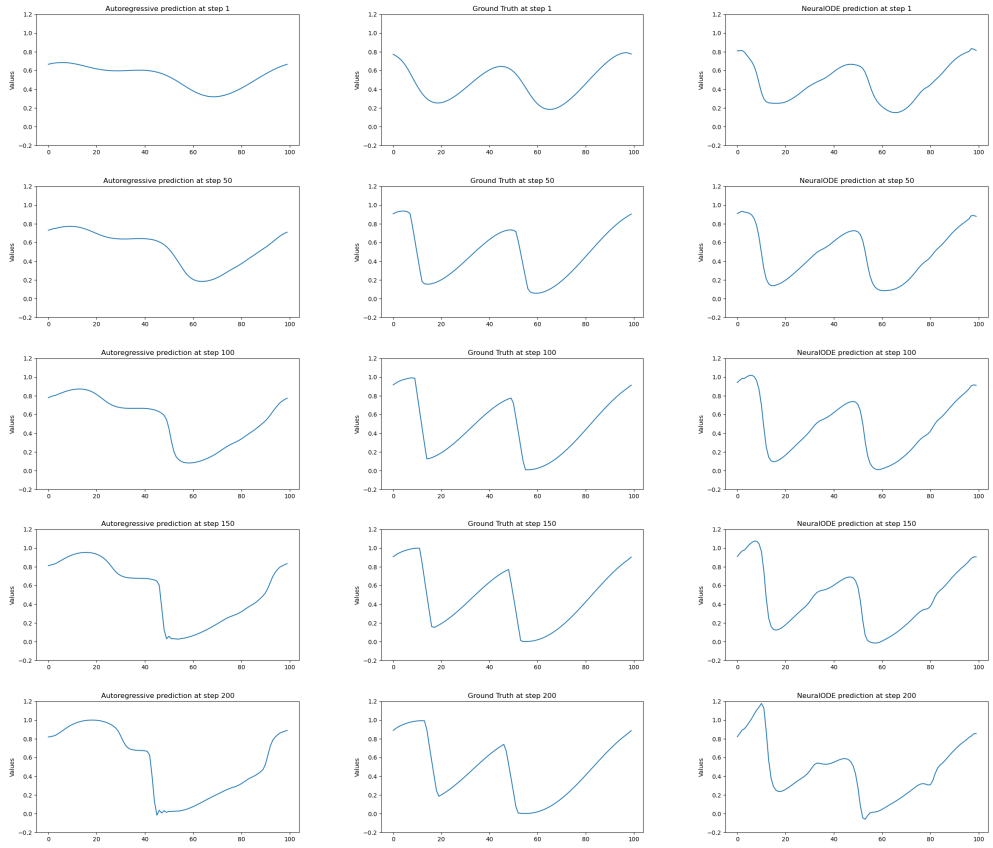


Figure 3: Visual comparison of the autoregressive baseline (left), ground truth (center), and Neural ODE predictions (right) across multiple timesteps (steps 1, 50, 100, 150, and 200).