University of California
Santa Barbara

Signature Transformations and Neural Differential Equations for Sequential Data Analysis

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Statistics and Applied Probability

by

Huiyu Jiang

Committee in charge:

Professor Tomoyuki Ichiba, Chair
Professor Jean-Pierre Fouque
Professor Ruimeng Hu

June 2024

The Dissertation of Huiyu Jiang is approved.

Professor Jean-Pierre Fouque

Professor Ruimeng Hu

Professor Tomoyuki Ichiba, Committee Chair

June 2024

Signature Transformations and Neural Differential Equations for Sequential Data Analysis

by

Huiyu Jiang

Acknowledgements

First, I extend my heartfelt gratitude to my advisor, Professor Ichiba. Throughout the long journey of my six-year doctoral program, I often felt overwhelmed and doubted whether I was suited for this academic path. Professor Ichiba not only paid attention to my mental health, but also consistently provided warm encouragement and support. He always listened patiently to my reports, which might have seemed naive, and offered valuable advice that helped me find my way forward. Without his comprehensive support and assistance, I cannot imagine being able to complete my studies successfully.

Secondly, I must thank my parents for their unwavering support behind the scenes, which allowed me to completely devote myself to my studies without distracting myself from daily trivialities. Being far from home, whenever I faced difficulties in life, I could always share them with my parents and other relatives, receiving comfort and encouragement in return. I am also grateful to their friends, such as Mr. Chen, Mr. Wang, Mr. Shen, and Mr. Xu, whose care and greetings often brought me warmth.

Lastly, I owe a debt of gratitude to my friends—Li, H., Yang, H., Wang, W., Mei, J., and Li, H.—whose companionship filled my monotonous life with laughter and joy. My collaboration with Luo, X. resulted in published articles, while Cheng, J., Li, Y., and Wang, S. provided numerous valuable suggestions and assistance in both my professional and personal life. All these people and experiences came together to help me overcome low points and complete my academic journey successfully.

Thank you to everyone who has helped me along the way. I hope for a bright future for us all.

Curriculum Vitæ

Huiyu Jiang

Education

2024	Ph.D. in Statistics, University of California, Santa Barbara.
2018	M.S. in Statistics, University of Wisconsin, Madison.
2017	B.S. in Statistics, Nanjing University

Publications

1. Luo, X., Wang, H., Huang, Z., **Jiang, H.**, Gangan, A., Jiang, S., & Sun, Y. (2024). Care: Modeling interacting dynamics under temporal environmental variation. Advances in Neural Information Processing Systems, 36.
2. Wang, H., **Jiang, H.**, Sun, J., Zhang, S., Chen, C., Hua, X. S., & Luo, X. (2023). DIOR: Learning to Hash With Label Noise Via Dual Partition and Contrastive Learning. IEEE Transactions on Knowledge and Data Engineering.
3. Luo, X., Gu, Y., **Jiang, H.**, Huang, J., Ju, W., Zhang, M., & Sun, Y. (2023). Graph ODE with Factorized Prototypes for Modeling Complicated Interacting Dynamics. arXiv preprint arXiv:2311.06554.
4. Luo, X., Yuan, J., Huang, Z., **Jiang, H.**, Qin, Y., Ju, W., ... & Sun, Y. (2023, July). Hope: High-order graph ode for modeling interacting dynamics. In International Conference on Machine Learning (pp. 23124-23139). PMLR.

Abstract

Signature Transformations and Neural Differential Equations for Sequential Data Analysis

by

Huiyu Jiang

Sequential data arise in numerous domains, including finance, Natural Language Processing, and healthcare. Effectively modeling and analyzing such data presents significant challenges due to their high dimensionality, nonstationarity, and complex dynamics. This dissertation tackles these challenges by using two powerful mathematical frameworks: signature transformations and neural differential equations (NDEs).

The first part explores the versatility of signature transformations in capturing the essential dynamics of sequential data, demonstrating their robustness and effectiveness for tasks such as classification, regression, and time series analysis. The second part investigates the practical applications of NDEs, including neural ordinary differential equations (NODEs) and neural stochastic differential equations (NSDEs), in modeling physical systems, biological processes, and other real-world phenomena with intricate dynamics. The third part delves into the efficacy of Channel Independent (CI) and Channel Dependent (CD) training strategies in multivariate time series forecasting. It presents a rigorous mathematical formulation and theoretical analysis to elucidate why the CI strategy often exceeds the CD strategy, particularly highlighting its robustness to distribution shifts between training and test datasets, a common scenario in real-world applications.

Through theoretical analysis, experimental evaluations, and case studies, this dissertation contributes to the advancement of sequential data analysis techniques. It also

provides a comprehensive understanding of signature transformations, NDEs, and the impacts of different training strategies, their properties, and their applications in various domains. The findings and methodologies presented in this work have the potential to impact a wide range of fields, allowing more accurate modeling, prediction, and decision-making processes involving sequential data.

Contents

Chapter 1

Introduction

1.1 Importance of Sequential Data Analysis

Time series analysis helps in identifying underlying patterns such as trends and seasonal variations in data over time. This is crucial in fields such as economics for understanding market trends, in meteorology for weather forecasting, and in retail for sales predictions. In addition, by understanding the normal patterns of time-dependent data, analysts can identify anomalies or outliers. This is particularly important in fraud detection in finance, fault detection in manufacturing processes, or unusual patient readings in healthcare. The importance of these analyses can be attributed to several key factors and applications.

1. Economic and Business Intelligence

    - **Financial Markets**: In finance, time series analysis is used to forecast stock prices [1, 2], exchange rates [3], and market trends. This helps traders and investors in making informed decisions, managing risks, and optimizing portfolios.

- **Supply Chain and Inventory Management**: Predicting future demand and sales to optimize inventory, reduce costs, and improve service levels [4, 5].

2. Scientific and Environmental Insights

   - **Astronomy**: Studying celestial objects' time series data to understand the universe's structure and dynamics [6].
   - **Environmental Science**: Monitoring climate patterns, pollution levels, and wildlife populations to predict changes and plan interventions [7].

3. Technological and Service Enhancements

   - **Speech Recognition and Natural Language Processing**: Analyzing sequential data to interpret human speech, improving virtual assistants and language translation services [8, 9, 10, 11, 12].
   - **Video Surveillance and Security**: Utilizing time series in video feeds for security, traffic management, and event detection [13].

4. Personalized Experiences and Services

   - **Recommendation Systems**: Employing user activity and preferences over time in technology platforms like shopping, entertainment, and social media to provide personalized recommendations, enhancing user experience and product promotion [14].
   - **Healthcare Tailoring**: Analysis of patient data over time leads to personalized treatment plans, predictive health insights, and improved healthcare services [15, 16].

5. Decision Making and Policy Development

- **Public Health and Epidemiology**: Understanding and predicting disease spread, evaluating intervention impacts, and informing policy making [17, 18].
- **Economic Indicators and Policy Making**: Analyzing and forecasting economic data to guide fiscal and monetary policies.
- **Resource Allocation in Energy and Utilities**: Predicting energy demand for efficient generation and distribution, ensuring sustainability [19].

These applications underscore the importance of time series and sequential data analysis in extracting meaningful insights, predicting future events, and making informed decisions.

1.2 Review of Sequential Data Analysis Methods

1. Traditional Time Series Analysis Methods

   Traditional models form the bedrock of time series analysis with their robust statistical foundations and decades of application across various fields:

   - **Autoregressive Integrated Moving Average (ARIMA)**: ARIMA models are among the most widely used forecasting methods, suitable for non-stationary series that can be made stationary by differencing. They combine autoregressive terms with moving averages to capture complex temporal structures [20, 21, 22].
   - **Exponential Smoothing (ES)**: These methods, including Holt-Winters, are used to smooth data and forecast in the presence of trends and seasonality, providing an intuitive approach to time series forecasting [23, 24, 25, 26].
   - **Seasonal Decomposition of Time Series (STL)**: STL is a versatile and robust method for decomposing a series into seasonal, trend, and residual com-

ponents, often used in conjunction with other forecasting methods to handle complex seasonal patterns [27, 28, 29].

- **Vector Autoregression (VAR)**: VAR models capture linear interdependencies among multiple time series, making them a popular choice in econometric forecasting and to understand the dynamics of the system [30, 31, 32, 33].
- **Dynamic Linear Models (DLM) / State Space Models**: These flexible models accommodate time-varying parameters and can incorporate both state and observation noise, widely used for robust filtering and forecasting [34, 35, 36, 37].

2. Jump Models and Counting Processes

   - **Poisson Process**: The Poisson process is a fundamental model for counting events over time, where events occur continuously and independently at a constant average rate [38].
   - **Hawkes Process**: An extension of the Poisson process, the Hawkes process models events that self-excite, meaning past events increase the likelihood of future events, useful in finance, social media analytics, and seismology [39, 40, 41].

3. Extending Traditional Models with Differential Equations Differential equations provide a theoretical framework for modeling dynamic systems and are integral to understanding changes in time series data:

   - **Ordinary Differential Equations (ODEs)**: ODEs model the rate of change in systems with deterministic behaviors and are widely used in engineering, physics, and biology [42, 43].

- **Partial Differential Equations (PDEs)**: PDEs extend to multiple variables and are crucial in representing physical phenomena, such as fluid dynamics, heat transfer, and electromagnetic fields [44].
- **Stochastic Differential Equations (SDEs)**: SDEs incorporate randomness directly into the evolution of the system, suitable for modeling more complex and unpredictable dynamics often found in financial markets and biological processes [45, 46, 47].

4. Modern Advances in Time Series Analysis

   The advent of deep learning has led to the development of innovative models that can capture intricate patterns and dependencies in sequential data:

   - **Recurrent Neural Networks (RNN) and Variants (LSTM, GRU)**: RNNs and their variants are specifically designed to handle sequential data, learning long-term dependencies and temporal patterns with significant success [48, 49, 50, 51, 52, 53, 54].
   - **Transformers**: Originally designed for natural language processing, Transformers have been adapted for sequential data, offering efficient handling of long-range dependencies [55, 56, 57, 58, 59, 60].

5. Differential Equations in Modern Data Analysis

   Bridging traditional differential equations with modern learning techniques has resulted in powerful models for sequential data:

   - **Neural Ordinary Differential Equations (NODEs)**: Integrating ODEs with neural networks, NODEs model the continuous evolution of data, offering a flexible approach to modeling complex systems [61, 62, 63, 64, 65, 66, 67].

- **Neural Stochastic Differential Equations (NSDEs)**: NSDEs extend NODEs by incorporating stochasticity, capturing the irregular and noisy nature of many real-world time series [68, 69, 70, 71].
- **Neural Controlled Differential Equations (NCDEs)**: NCDEs specifically address the challenges of irregularly sampled time series data, providing a robust framework for handling a wide variety of data types [72, 73, 74].

1.3 Current Trends and Challenges

Recent advances have significantly enhanced the capability of time series models, but they also present new challenges and opportunities:

- Handling Non-Stationarity: Non-stationary data, common in real-world applications, pose significant challenges. Models must adapt to changing trends and variances over time. Techniques for tackling non-stationarity include differencing, transformation, and advanced decomposition methods [75].
- High-Dimensionality and Multivariate Time Series: As data complexity increases, models must efficiently handle multivariate and high-dimensional time series, capturing interactions between multiple series and variables [76].
- Irregular Sampling and Missing Values: Real-world data often come with irregularly sampled or with missing values. Models must be robust to these irregularities, often requiring innovative approaches to interpolation and imputation [77].
- Interpretability and Trustworthiness: Although deep learning models provide improved predictive accuracy, their "black-box" nature poses challenges to interpretability and trust. There is a growing demand for models that balance predictive power with explainability, especially in critical applications [78].

- Computational Efficiency and Scalability: With the increasing size and complexity of the data, computational efficiency and the ability to scale become critical. This includes the need for models that can be trained and deployed efficiently without compromising performance [28].

In response to these challenges, this thesis presents a comprehensive study and novel contributions in the following structure:

- **Chapter 2: Applications of Signature Transformation**: Focusing on the versatility of signature transformation in capturing essential dynamics of sequential data, this chapter will delve into its application in classification, distribution regression, and beyond, providing a robust solution for complex time series analysis.

- **Chapter 3: Applications of Neural Differential Equations**: Exploring the practical applications of NODEs and NSDEs, this chapter will demonstrate their efficacy in modeling physical systems and other real-world phenomena, showcasing the power of continuous-depth models.

- **Chapter 4: Training Strategy Analysis - Channel Independence versus Channel Dependence**: This chapter investigates the efficacy of Channel Independent (CI) and Channel Dependent (CD) training strategies in multivariate time series forecasting. It presents a rigorous mathematical formulation and theoretical analysis to elucidate why the CI strategy often exceeds the CD strategy. Our analysis reveals that the superiority of the CI approach is mainly due to its robustness to distribution shifts between training and test datasets, a common scenario in real-world applications.

Chapter 2

Rough Path Theory and Signature

2.1 Rough Path Theory

Rough Path Theory emerges as a pivotal mathematical framework designed to analyze and integrate paths characterized by highly irregular behaviors, thus extending classical analysis to signals or paths that are too "rough" for traditional methods. This chapter delves into the foundational concepts of Rough Path Theory, including the basics of rough paths with formal definitions and preliminary notations that underpin this advanced field. The theory extends the boundaries of conventional analysis, offering new vistas for understanding complex, non-smooth dynamics that are prevalent in numerous real-world phenomena. For readers keen on exploring this topic in greater depth, works like [79, 80, 81, 82, 83, 84, 85] provide comprehensive insights into the intricacies of Rough Path Theory. These references serve as a cornerstone for those wishing to gain a thorough grasp of the theory's principles and applications, offering a blend of foundational knowledge and advanced theoretical developments.

2.1.1 Preliminaries and Notations

Before diving into the specifics of rough paths and their properties, let's establish the basic notations and concepts used throughout the discussion of Rough Path Theory.

Let $I = [0, T]$ be a closed time interval and consider a path $X : I \to \mathbb{R}^d$ mapping from the interval into a d-dimensional space. The set of all partitions of I is denoted as $\mathcal{P}(I)$, where a partition is a finite sequence $D := (t_0, t_1, \ldots, t_n)$ such that $0 = t_0 < t_1 < \ldots < t_n = T$.

Furthermore, we introduce the concept of the increment of a path over a subinterval of I. For any $[s, t] \subset I$, the increment of X over $[s, t]$ is given by $X_{s,t} = X_t - X_s$, where X_s and X_t are the values of the path at times s and t, respectively.

2.1.2 Basics of Rough Paths

We first introduce the concept of p-variation, a measure of a path's irregularity, which plays a central role in defining rough paths.

Definition 1 (p-variation)**.** The p-variation of a path X on the interval $I = [0, T]$ is defined as:

$$V_p(X; [0, T]) = \left(\sup_{D \in \mathcal{P}(I)} \sum_{i=0}^{n-1} \|X_{t_{i+1}} - X_{t_i}\|^p \right)^{\frac{1}{p}}, \tag{2.1}$$

where $\|\cdot\|$ denotes the Euclidean norm in $\mathbb{R}^d$. A path is said to have finite p-variation if $V_p(X; [0, T])$ is finite for some $p \geq 1$.

The concept of p-variation is foundational in understanding the irregularity and complexity of paths considered in rough path theory. It provides a quantifiable measure of a path's roughness and is a prerequisite for defining rough paths [79]. The choice of p directly correlates to the regularity of the path; higher p values indicate higher irregularity, and for a path with finite p-variation, it can be shown that it is continuous with

respect to the p-variation norm, a property fundamental when dealing with integration and differential equations driven by such paths [84].

Definition 2 (Tensor Product)**.** Given vectors $v_1, v_2, \ldots, v_n \in \mathbb{R}^d$, their tensor product $v_1 \otimes v_2 \otimes \ldots \otimes v_n$ is an element in $(\mathbb{R}^d)^{\otimes n}$, defined by the outer product of the vectors. The result is a tensor of order n, which can be represented as a multidimensional array.

For example, consider two vectors $v_1, v_2 \in \mathbb{R}^2$ where $v_1 = (a_1, a_2)$ and $v_2 = (b_1, b_2)$. Their tensor product, $v_1 \otimes v_2$, is computed as follows:

$$v_1 \otimes v_2 = \begin{pmatrix} a_1 \cdot b_1 & a_1 \cdot b_2 \\ a_2 \cdot b_1 & a_2 \cdot b_2 \end{pmatrix}.$$

This result is a 2×2 matrix, which is an element of $(\mathbb{R}^2)^{\otimes 2}$. The matrix entries are formed by multiplying the elements of v_1 with those of v_2 in a manner consistent with the definition of the outer product.

The tensor product of vectors leads to higher-dimensional analogs of vectors, known as tensors. The role of tensor algebra is foundational in Rough Path Theory, providing the necessary structure for understanding iterated integrals and signatures. The space of tensors, equipped with shuffle and unshuffle products, forms an algebra that is deeply intertwined with the combinatorial aspects of iterated integrals.

With this understanding, we can now define a rough path.

Definition 3 (Rough Path)**.** A rough path over a path $X : I \to \mathbb{R}^d$ is a sequence $\mathbf{X} = (1, X, \mathbb{X}^{(2)}, \ldots, \mathbb{X}^{(n)})$ where X is the path itself, and $\mathbb{X}^{(i)}$ for $i \geq 2$ are the iterated integrals of X. These iterated integrals are defined inductively by:

$$\mathbb{X}^{(n)}_{s,t} = \int_s^t \mathbb{X}^{(n-1)}_{s,u} \otimes dX_u. \tag{2.2}$$

This sequence must satisfy specific algebraic and analytic conditions, including continuity with respect to the p-variation norm and a coherence condition relating the iterated integrals. A rough path is thus an enhanced version of the traditional path, incorporating not only the path's increments but also higher-order information about its behavior, allowing for the analysis and integration of paths with irregular trajectories.

These preliminary concepts lay the foundation for understanding Rough Path Theory and its applications. The p-variation provides a means to quantify the irregularity of paths, while the rough path itself, with its iterated integrals, offers a robust framework for analyzing and integrating such paths.

2.1.3 Rough Integrals

In rough path theory, controlled paths constitute a specific class of paths that align with a given rough path, thereby enabling meaningful integration against it.

Definition 4 (Controlled Path)**.** A path $Y : [0, T] \to \mathbb{R}^e$ is said to be controlled by a rough path $\mathbf{X} = (1, X, \mathbb{X}^{(2)}, \ldots, \mathbb{X}^{(n)})$ over the interval $[0, T]$ if there exists a Gubinelli derivative $Y' : [0, T] \to (\mathbb{R}^d)^{\otimes(n-1)}$, which approximates the increments of Y in relation to the increments of $\mathbf{X}$. Specifically, the remainder term $R_{s,t}$ is defined as

$$R_{s,t} = Y_t - Y_s - Y'_s \otimes X_{s,t},$$

where $R_{s,t}$ must exhibit finite p-variation for some $p \geq 1$, indicating its negligible effect over small intervals. Here, $\otimes$ denotes the tensor product operation between Y' and $X_{s,t}$.

This definition ensures that the path Y closely follows the structure of the driving rough path $\mathbf{X}$, thereby facilitating the rough integral of Y against $\mathbf{X}$. This extends classical integration to include paths with lower regularity.

Definition 5 (Rough Integral)**.** Given a controlled path Y, controlled by a rough path $\mathbf{X}$, the rough integral of Y with respect to $\mathbf{X}$ over an interval $I = [0, T]$ is defined as

$$\int_0^T Y_u \, d\mathbf{X}_u = \lim_{\|D\| \to 0} \sum_{[t_i, t_{i+1}] \in D} Y_{t_i} \mathbf{X}_{t_i, t_{i+1}}, \tag{2.3}$$

where D represents a partition of $[0, T]$. Here, $\|D\| = \max_i(t_{i+1} - t_i)$ denotes the maximum length of the subintervals.

The definitions provide a solid mathematical foundation, setting the stage for exploring system evolution under the influence of rough paths.

The concept of rough integrals, introduced by Terry Lyons, broadens the scope of classical integration to include semimartingales and functions beyond bounded variation. This breakthrough permits the integration of paths that exhibit merely Hölder continuity with an exponent greater than $\frac{1}{2}$. As a result, rough path theory, and by extension, rough integrals, empower the meticulous examination of differential equations driven by such paths.

2.1.4 Controlled Differential Equations (CDEs)

Controlled Differential Equations (CDEs) extend classical differential equations to accommodate driving signals that are rough paths, marking a significant advancement in Rough Path Theory. This extension facilitates a deeper understanding of system evolution under the influence of irregular paths, addressing the complexities of modeling real-world phenomena.

Definition 6 (Controlled Differential Equations)**.** A Controlled Differential Equation

(CDE), driven by a rough path $\mathbf{X}$, is formulated as:

$$dY_t = V(Y_t)\, d\mathbf{X}_t, \tag{2.4}$$

where $Y : [0, T] \to \mathbb{R}^e$ denotes the system's state variable, and $V : \mathbb{R}^e \to L(\mathbb{R}^d, \mathbb{R}^e)$ represents a smooth vector field, dictating the system's response. The differential $d\mathbf{X}_t$ symbolizes integration against the rough path $\mathbf{X}$. The solution Y_t evolves in a manner controlled by the rough path $\mathbf{X}$, reflecting the cumulative effect of the driving signal on the system

Properties

The solution to a CDE is a controlled path that adheres to the prescribed dynamics. Formally, given an initial condition Y_0, the solution Y to the CDE over $[0, T]$ is the controlled path satisfying:

$$Y_t = Y_0 + \int_0^t V(Y_s) d\mathbf{X}_s,$$

where the integral is understood as a rough integral. The existence, uniqueness, and stability of solutions typically hinge upon the regularity of the vector field V and the rough path $\mathbf{X}$, along with certain growth and Lipschitz conditions [84].

CDEs generalize several core principles from classical differential equations to the context of rough paths, including:

1. **Existence and Uniqueness**: Provided specific conditions on the driving rough path $\mathbf{X}$ and the vector field V are met, a unique solution to the CDE exists for any given initial condition [79].

2. **Continuity**: The solutions to CDEs exhibit continuous dependence on both the initial conditions and the driving rough path. This continuity guarantees that mall

changes in the input lead to small changes in the output [82].

Controlled Differential Equations are pivotal in various domains, especially in modeling complex systems influenced by irregular signals. They are extensively used in financial mathematics for modeling market dynamics where they can represent the evolution of asset prices or interest rates under stochastic influences, in engineering for signal processing, and in many other fields where inputs are inherently noisy or irregular. Specifically, they pave the way for sophisticated models like neural CDEs, which are discussed in Chapter 3. These models provide powerful frameworks for understanding and predicting the behavior of complex systems under irregular influences.

2.2 Signature Transform

2.2.1 Mathematical Definitions

The signature of a path offers a detailed summary by capturing its geometric properties through an infinite series of iterated integrals.

Definition 7. Let E be a Banach space with a metric $d(\cdot, \cdot)$. The space of formal series of tensors over E, denoted by $T(E)$, is defined as

$$T(E) := \{\mathbf{a} = (a_0, a_1, \cdots) | \forall n \in \mathbb{N}, a_n \in E^{\otimes n}\}. \tag{2.5}$$

Here, $\otimes$ represents the tensor product, with the convention $E^{\otimes 0} = \mathbb{R}$.

For elements $\mathbf{a} = (a_0, a_1, \cdots)$ and $\mathbf{b} = (b_0, b_1, \cdots) \in T(E)$, their addition and tensor

product are defined as follows:

$$\mathbf{a} + \mathbf{b} = (a_0 + b_0, a_1 + b_1, \cdots);$$
$$\mathbf{a} \otimes \mathbf{b} = (c_0, c_1, \cdots),$$

where $c_k = \sum_{j=0}^{k} a_j \otimes b_{k-j}$ computes the k-th convolution from the first k elements of $\mathbf{a}$ and $\mathbf{b}$.

Definition 8 (Signature)**.** For a continuous path $X(\cdot) : I \to E$ of finite p-variation, $p < 2$, its signature, $S(X)_I$, is defined as

$$S(X)_I := (1, X_I^1, X_I^2, \cdots) \in T(E), \tag{2.6}$$

with each k-th level signature X_I^k being the k-th iterated integral:

$$X_I^k = \int_{s<u_1<\cdots<u_k<t} dX_{u_1} \otimes \cdots \otimes dX_{u_k},$$

for $k \geq 1$.

Considering $E = \mathbb{R}^m$ for an m-dimensional path $X(t) = (X^1(t), \cdots, X^m(t))$, we express the signature for a multi-index $(i_1, \cdots, i_k)$, where $i_1, \cdots, i_k \in \{1, \cdots, m\}$, as:

$$S(X)_{s,t}^{(i_1,\cdots,i_k)} = \int_{s<u_1<\cdots<u_k<t} dX_{u_1}^{i_1} \cdots dX_{u_k}^{i_k}. \tag{2.7}$$

Thus we have the signature of X as a vector:

$$S(X)_I = (1, S(X)_I^{(1)}, \cdots, S(X)_I^{(m)}, S(X)_I^{(1,1)}, \cdots, S(X)_I^{(1,m)}, \cdots) \in T(\mathbb{R}^m).$$

Definition 9 (Truncated signature)**.** The signature of path X over interval I truncated

at level l is

$$S^l(X)_I = (1, S(X)_I^{(1)}, \cdots, S(X)_I^{(m)}, \cdots, S(X)_I^{\overbrace{(m, \cdots, m)}^{l}}). \tag{2.8}$$

2.2.2 Properties of the Signature

The signature of a path has several key properties that underscore its utility in analyzing path data:

1. **Shift Invariance:** Suppose we shift the whole path along a given direction, like $\tilde{X}(\cdot) = X(\cdot) + \vec{v}$ where $\vec{v} = (v_1, \cdots, v_m) \in \mathbb{R}^m$, then $S(\tilde{X})_I = S(X)_I$ because $\forall i_1, \cdots, i_k \in \{1, \cdots, m\}$,

$$S(\tilde{X})_{s,t}^{(i_1,\cdots,i_k)} = \int_{s<u_1<\cdots<u_k<t} d(X_{u_1}^{i_1} - v_{i_1}) \cdots d(X_{u_k}^{i_k} - v_{i_k}) = S(X)_{s,t}^{(i_1,\cdots,i_k)}. \tag{2.9}$$

2. **Invariance under time reparametrisations:** Suppose $\psi : I = [0, T] \to [0, T]$ is a reparametrisation and $\tilde{X}(t) = X_{\psi(t)}$, then we have $\dot{\tilde{X}}_t^i = \dot{X}_{\psi(t)}^i \dot{\psi}(t), \forall i \in \{1, \cdots, m\}$. From this we have $\forall i, j \in \{1, \cdots, m\}$,

$$S(\tilde{X})_I^{(i,j)} = \int_0^T \tilde{X}_t^i d\tilde{X}_t^j = \int_0^T X_{\psi(t)}^i \dot{X}_{\psi(t)}^j \dot{\psi}(t) dt = \int_0^T X_t^i dX_t^j = S(X)_I^{(i,j)}. \tag{2.10}$$

   By induction, we can show that $S(\tilde{X})_I = S(X)_I$.

   Here is an example. Suppose the original path $X(t) = \begin{pmatrix} t \\ t^2 \end{pmatrix}, \quad t \in [0, 1]$, and the reparameterization function $\psi(t) = t^3$. Then we have the reparameterized path $\tilde{X}(t) = \begin{pmatrix} t^3 \\ t^6 \end{pmatrix}, \quad t \in [0, 1]$.

Now let's calculate the Signatures. For the original path X, we have

$$S(X)_{(1)} = \int_0^1 dX(t) = \begin{pmatrix} \int_0^1 dt \\ \int_0^1 2t\, dt \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix},$$

$$S(X)_{(1,1)} = \int_0^1 \int_0^s dX(u) \otimes dX(s) = \int_0^1 \int_0^s \begin{pmatrix} 1 \\ 2u \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 2s \end{pmatrix} du\, ds$$

$$= \int_0^1 \int_0^s \begin{pmatrix} 1 \\ 2s \\ 2u \\ 4us \end{pmatrix} du\, ds = \begin{pmatrix} \frac{1}{2} \\ \frac{2}{3} \\ \frac{1}{3} \\ \frac{1}{2} \end{pmatrix}.$$

For the reparameterized Path $\tilde{X}(t)$, we have

$$S(\tilde{X})_{(1)} = \int_0^1 d\tilde{X}(t) = \int_0^1 \begin{pmatrix} 3t^2 \\ 6t^5 \end{pmatrix} dt = \begin{pmatrix} \int_0^1 3t^2\, dt \\ \int_0^1 6t^5\, dt \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix},$$

$$S(\tilde{X})_{(1,1)} = \int_0^1 \int_0^s d\tilde{X}(u) \otimes d\tilde{X}(s) = \int_0^1 \int_0^s \begin{pmatrix} 3u^2 \\ 6u^5 \end{pmatrix} \otimes \begin{pmatrix} 3s^2 \\ 6s^5 \end{pmatrix} du\, ds$$

$$= \int_0^1 \int_0^s \begin{pmatrix} 9u^2s^2 \\ 18u^2s^5 \\ 18u^5s^2 \\ 36u^5s^5 \end{pmatrix} du\, ds = \begin{pmatrix} \frac{1}{2} \\ \frac{2}{3} \\ \frac{1}{3} \\ \frac{1}{2} \end{pmatrix}.$$

After calculating these integrals, we can see the signatures of the original path $X(t)$ and the reparameterized path $\tilde{X}(t)$ are equal:

$$S(\tilde{X})_I = S(X)_I \text{ for all } I.$$

This demonstrates the invariance of the signature under time reparameterization.

Chen's Identity and **Uniqueness of Signature** are fundamental theorems that further elucidate the mathematical structure and the robustness of the signature as an analytical tool:

Theorem 1 (Chen's Identity)**.** For continuous paths $X(\cdot) : [0, t] \to E$ and $Y(\cdot) : [t, T] \to E$ of finite p-variation ($p < 2$), their concatenation $(X * Y)(u)$, defined as

$$(X * Y)(u) = \begin{cases} X(u), & \text{if } u \in [0, t], \\ X(t) + Y(u) - Y(t), & \text{if } u \in [t, T], \end{cases}$$

satisfies

$$S(X * Y) = S(X) \otimes S(Y). \tag{2.11}$$

Applying Chen's identity, we can calculate the k-th level signature of path by

$$X^k_{[s,t]} = \sum_{i=1}^{n} X^k_{[t_{i-1},t_i]} + \sum_{j=1}^{k-1} \sum_{i=1}^{n} X^j_{[s,t_{i-1}]} \otimes X^{k-j}_{[t_{i-1},t]}, \tag{2.12}$$

for every subinterval $[s, t] \subseteq I$ with a given partition $s = t_0 < t_1 < \cdots < t_n = t$.

Theorem 2 (Uniqueness of Signature [86])**.** For any two paths $X, Y : [0, T] \to \mathbb{R}^d$ of finite p-variation, if their signatures are identical over the interval $[0, T]$, then X and Y are identical up to tree-like equivalence, assuming neither path is tree-like.

This theorem underscores the significance of the signature transform in ensuring that the encoded geometric information of a path is sufficiently rich to recover the path's original trajectory, excluding non-informative tree-like segments. The proof of this theorem involves sophisticated mathematical concepts from rough path theory, please see [83] for a detailed exposition of the proof and further discussion.

2.2.3 Log Signature: A Compact Representation

The log signature provides a compact representation of a path's signature, which is particularly useful when dealing with high-dimensional data. It simplifies the signature's complexity by reducing the number of terms needed to capture the path's essential information.

Given a path $X(\cdot) : I \to \mathbb{R}^m$ and its signature truncated at level l, the exponential growth of the signature vector's length, $\text{len}(S^l(X)_I)$, can be problematic for model training due to high dimensionality:

$$\text{len}(S^l(X)_I) = \sum_{i=0}^{l} m^i = \frac{m^{l+1} - 1}{m - 1}.$$

For instance, augmenting a one-dimensional stochastic process $X(\cdot) : I \to \mathbb{R}$ to include time as $\tilde{X}(t) = (t, X(t)),\ t \in I$, with $m = 2$. Then its length of the signature truncated at level l is $2^{l+1} - 1$. If the length of observed path is $n = 100$, we can see that when $l \geq 6$, we have $\text{len}(S^l(X)_I) > n$. It means we use more features than the length of the path to train the model. This is because we use redundant information in the signature vector. A typical example is that

$$S(X)_I^{\overbrace{(2, \cdots, 2)}^{l}} = \frac{(X^2(T) - X^2(0))^l}{l!}.$$

Thus we hope to store fewer values to improve the efficiency of training models.

Now, for fixed N, denote $\mathbf{t}^N(\mathbb{R}^m) = \{\mathbf{a} \in T((\mathbb{R}^m)) | a = (0, a_1, a_2, \cdots),\ \forall n > N, a_n = 0\}$. Then for the vector space $(\mathbf{t}^N(\mathbb{R}^m), +, .)$, we define the commutator:

$$[g, h] := g \otimes h - h \otimes g \in \mathbf{t}^N(\mathbb{R}^m),\ \forall g, h \in \mathbf{t}^N(\mathbb{R}^m). \tag{2.13}$$

It is easy to check this bilinear map is anticommutative:

$$[g,h] = -[h,g],\ \forall g,h \in \mathbf{t}^N(\mathbb{R}^m), \tag{2.14}$$

and it satisfies the Jacobi identity:

$$[g,[h,k]] + [h,[k,g]] + [k,[g,h]] = 0,\ \forall g,h,k \in \mathbf{t}^N(\mathbb{R}^m). \tag{2.15}$$

Thus, we have $(\mathbf{t}^N(\mathbb{R}^m), +, ., [\cdot,\cdot])$ is a Lie algebra.

Definition 10 (Exponential and logarithm map)**.** Let $\mathbf{a} = (0, a_1, \cdots) \in \mathbf{t}^N(\mathbb{R}^m)$, then the exponential map is defined by:

$$\exp(\mathbf{a}) = 1 + \sum_{n=1}^{N} \frac{\mathbf{a}^{\otimes n}}{n!} \in 1 + \mathbf{t}^N(\mathbb{R}^m). \tag{2.16}$$

The logarithm map is defined by:

$$\log(1+\mathbf{a}) = \sum_{n=1}^{N} \frac{(-1)^{n-1}}{n} \mathbf{a}^{\otimes n} \in \mathbf{t}^N(\mathbb{R}^m). \tag{2.17}$$

Let $X(\cdot) :\ I \to E$ be a continuous path of finite p-variation for some $p < 2$. Then the log signature of the path X truncated at level l is defined by: $l\mathcal{S}^l(X)_I = \log(S^l(X)_I)$.

For example, suppose $\mathbf{a}, \mathbf{b} \in \mathbf{t}^3(\mathbb{R}^m)$, then

$$\begin{aligned}
\exp(\mathbf{a}) \otimes \exp(\mathbf{b}) =& \Big(1 + \mathbf{a} + \frac{\mathbf{a}^{\otimes 2}}{2} + \frac{\mathbf{a}^{\otimes 3}}{6}\Big) \otimes \Big(1 + \mathbf{b} + \frac{\mathbf{b}^{\otimes 2}}{2} + \frac{\mathbf{b}^{\otimes 3}}{6}\Big) \\
=& 1 + \mathbf{a} + \mathbf{b} + \frac{\mathbf{a}^{\otimes 2}}{2} + \mathbf{a}\otimes\mathbf{b} + \frac{\mathbf{b}^{\otimes 2}}{2} \\
&+ \frac{\mathbf{a}^{\otimes 3}}{6} + \frac{\mathbf{b}^{\otimes 3}}{6} + \frac{\mathbf{a}^{\otimes 2}\otimes\mathbf{b}}{2} + \frac{\mathbf{a}\otimes\mathbf{b}^{\otimes 2}}{2}, \\
\Rightarrow \log(\exp(\mathbf{a})\otimes\exp(\mathbf{b})) =& \mathbf{a} + \mathbf{b} + \frac{1}{2}[\mathbf{a},\mathbf{b}] + \frac{1}{12}[\mathbf{a},[\mathbf{a},\mathbf{b}]] + \frac{1}{12}[\mathbf{b},[\mathbf{b},\mathbf{a}]].
\end{aligned}$$

Theorem 3. The log signature of path $X(\cdot): I \to \mathbb{R}^m$ truncated at level l has length:

$$\text{len}(lS^l(X)_I) = \sum_{i=1}^{l} \frac{1}{i} \sum_{x|i} \mu(\frac{i}{x}) m^x, \tag{2.18}$$

where μ is the Mobius function. Detailed proof can be found in [87].

From this theorem we have the conclusion that the length of log signature vector is less than that of a signature for the same path at the same truncation level. In [88], there is a detailed table showing how log signature transformation significantly reduces the number of features as level increases. Based on this property, we can use log signature transformation to train the models based on fewer features without reducing much precision rate.

2.2.4 Signature Calculation

Given a one-dimensional time series, the calculation of its signature requires the transformation of the series into a multi-dimensional object that the signature transform can process. Two common methods for this transformation are the addition of a time dimension and the lead-lag transformation.

- **Adding Time Dimension**: The addition of a time dimension involves augmenting each observation X_t in the time series with its corresponding timestamp t, effectively transforming the series into a two-dimensional path. Mathematically, this can be represented as:

  $$\widetilde{X}_t = (t, X_t),$$

  where $\widetilde{X}_t$ denotes the augmented path. This transformation ensures that the resulting path captures not only the values of the time series but also the temporal ordering of these values, which is crucial for the signature calculation.

- **Lead-Lag Transformation** Given a time series $\{X_t\}_{t=1}^{n}$, the lead-lag transformation constructs a new sequence by interleaving the original series with its own lagged version, resulting in a two-dimensional path. This transformation can be concisely expressed as:

$$\text{Lead-Lag}(X) = \{(X_t, X_t), (X_{t+1}, X_t)\}_{t=1}^{n-1} \cup \{(X_n, X_n)\}. \tag{2.19}$$

  This method ensures that each point in the time series is represented alongside its preceding value, enriching the dataset without favoring either the lead or lag values.

In practice, the lead-lag transformation may offer advantages over simply adding a time dimension, particularly when the magnitude of the time series data significantly differs from the scale of the time values. In the case of time addition, $\widetilde{X}_t = (t, X_t)$, such disparity in scale could result in one dimension overwhelming the other, potentially obscuring important information. The lead-lag transformation, by contrast, preserves the scale of the original data and avoids this issue by maintaining consistency in the dimensions' magnitude. This characteristic makes the lead-lag transformation particularly suitable for datasets where preserving the relative scale of data points is crucial.

For computational purposes, packages such as *esig* [1], *iisignature* [89], and *signatory* [90] are available. Among these, *signatory* is preferred for its compatibility with PyTorch, enabling GPU acceleration and efficient tensor operations, which are advantageous for deep learning models.

[1] available from https://pypi.org/project/esig/

2.3 Methods

2.3.1 Universal Approximation

A cornerstone of rough path theory is the Universal Approximation Theorem [91], which posits that for any continuous function f over paths in a compact set, there exists a linear combination of signature elements that can approximate f within any given error bound. Formally, it states:

Theorem 4 (Universal Approximation Theorem for Signatures [91])**.** Let $\mathcal{X} \subset \mathcal{C}(I, E)$ be a compact set of paths and consider a continuous function $f : \mathcal{X} \to \mathbb{R}$. Then for any $\epsilon > 0$ there exists a truncation level $n \geq 0$ such that for any path $X \in \mathcal{X}$,

$$\left| f(X) - \sum_{k=0}^{n} \sum_{J \in \{1,\dots,d\}^k} \alpha_J S(X)^J \right| < \epsilon, \tag{2.20}$$

where $\alpha_J \in \mathbb{R}$ are scalar coefficients.

The Universal Approximation Theorem highlights the signature's ability to extract and represent the critical information of paths, facilitating the accurate approximation of path-dependent functions. This characteristic makes signatures immensely useful for feature generation from sequential data, allowing the application of diverse machine learning and deep learning methodologies to develop predictive models. This approach leverages the rich information encoded in the signatures, optimizing model performance across various data analysis tasks.

2.3.2 Missing data and augmentation

In practical scenarios, datasets often come with missing observations. Traditional approaches might discard such incomplete data, potentially losing valuable information.

To maximize data utilization, various methods are employed to impute missing parts, enhancing model performance.

The signature method exhibits remarkable robustness to missing data, a property particularly beneficial for handling real-world datasets. Consider a path $X(t) = (t, B_t)$ over $I = [0, 1]$, with B_t representing standard Brownian Motion. The signature elements $S(X)_I^{(1,2)}$ and $S(X)_I^{(2,1)}$ are calculated as follows:

$$S(X)_I^{(1,2)} = \int_{0<s<t<1} dX_s^1 dX_t^2 = \int_0^1 t dB_t; \quad S(X)_I^{(2,1)} = \int_0^1 B_t dt.$$

These calculations are visually represented in Figure 2.1, where the shaded areas under the curve illustrate the integral's computation.

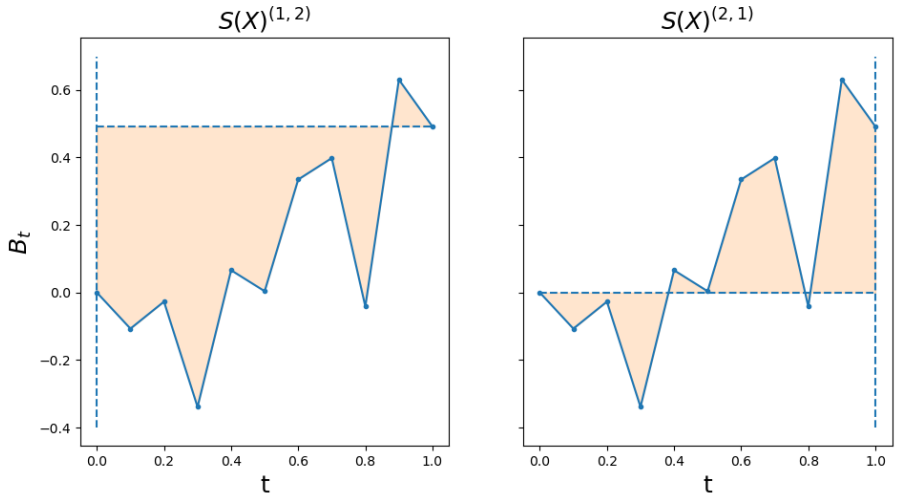
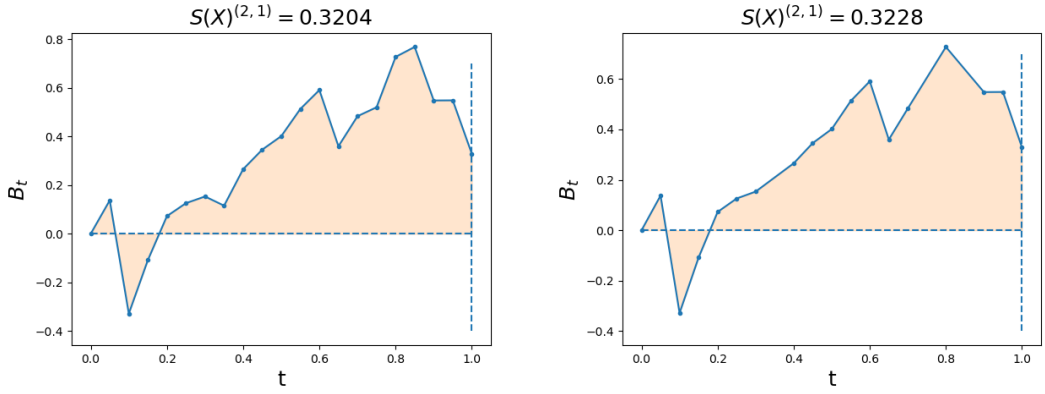


Figure 2.1: An illustration of the elements from the signature of the path. (Left) The shadow area represents $S(X)_I^{(1,2)}$. (Right) The shadow area represents $S(X)_I^{(2,1)}$.

The robustness of the signature method to missing data is evident when considering the removal of a data point from the path. Despite the absence of a point, the integral calculations and thus the signature elements $S(X)_I^{(1,2)}$ and $S(X)_I^{(2,1)}$ remain largely unaffected. This resilience is illustrated by the negligible shaded areas in Figure 2.2, even

when some data points are omitted. The integral-based nature of the signature inherently accounts for the continuous structure of the path, minimizing the impact of missing discrete observations. This property is crucial for real-world applications where data may often be incomplete. By leveraging the signature method's inherent robustness to missing data, we can retain more information from the available data, thereby enhancing model training and analysis without substantial loss of accuracy or fidelity.



(a) Full path (b) Signature comparison

Figure 2.2: An illustration of the robustness of signature transformation.

Despite the signature method's robustness to missing data, enhancing the dataset through strategic transformations can further improve model performance. It's essential to recognize that the significance of missing data often lies not in the missing values themselves but in understanding the circumstances of their absence. For instance, missing stock prices might not be as interesting as the dates when these values are missing. The absence of values often coincides with market closures during festivals or, more critically, when a circuit breaker is activated. Similarly, in medical studies, if a treatment causes discomfort, patients might cease recording specific indices in the research. Such patterns of missingness are non-random and warrant further investigation. To address this, we might introduce an extra dimension to capture information about the occurrence of

missing values.

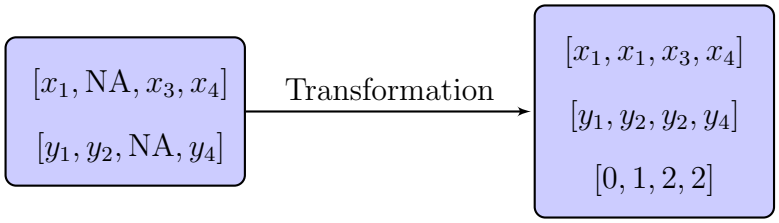


Figure 2.3: Data augmentation example for handling missing values.

Figure 2.3 illustrates an example of data augmentation for missing values. Here, **NA** represents the missing values in the original dataset. We apply two transformations to augment the data and mitigate the impact of missing values. Firstly, we replace **NA** values with the nearest preceding observations in the dataset, maintaining the original length of the dataset. This approach ensures that we preserve the non-**NA** observations at each time point, thereby retaining significant information without significantly altering the signature elements, due to the inherent robustness of the signature method. Secondly, we introduce an additional dimension to record the cumulative count of missing values at each time point. This dimension provides critical insights into the frequency and patterns of missingness. Through this augmentation process, we generate a new dataset which, upon applying the signature transformation, yields more features. Given that $\text{len}(S^l(X)_I) = \frac{m^{l+1}-1}{m-1}$, these additional features can significantly enhance the model's performance.

2.3.3 Distribution regression

Suppose we have a collection of observations structured in input-output pairs:

$$\left(\left\{X^{1,j} : I \to E\right\}_{j=1}^{N_1}, y^1 \in \mathbb{R}\right), \ldots, \left(\left\{X^{m,j} : I \to E\right\}_{j=1}^{N_m}, y^m \in \mathbb{R}\right),$$

where there are m groups of observations. For each group i, the output y^i is a fixed real number, and $\{X^{i,j}\}_{j=1}^{N_i}$ represents N_i independent paths corresponding to the parameter y^i for $i = 1, \ldots, m$. By incorporating the time parameter, we represent each path as $X^{i,j} = \left\{(t_1, X_1^{i,j}), \ldots, (t_{l_{i,j}}, X_{l_{i,j}}^{i,j})\right\}$, where $l_{i,j}$ is the length of the j-th sample path in group i.

The empirical measure of group i is defined as:

$$\delta_i = \frac{1}{N_i} \sum_{j=1}^{N_i} \delta_{X^{i,j}} \in \mathcal{P}(\mathcal{X}), \tag{2.21}$$

where $\delta_{X^{i,j}}$ denotes the Dirac measure with a point mass at path $X^{i,j}$. The expected signature of the empirical measure is then given by:

$$\Phi(\delta_i) = \frac{1}{N_i} \sum_{j=1}^{N_i} S(X^{i,j}), \tag{2.22}$$

illustrating the foundational approach for two distribution regression methods: kernel-based and pathwise.

- A kernel-based approach (**kerES**):

  (1) Calculate the signature of each path $X^{i,j}$ truncated at a fixed level l.

  (2) Average the signature of paths from the same group to get the expected signature, i.e, $\Phi^l(\delta^i) := \frac{1}{N_i} \sum_{j=1}^{N_i} S^l(X^{i,j})$.

  (3) Apply kernel ridge regression method to the transformed data $\{(\Phi^l(\delta^1), y^1), \cdots, (\Phi^l(\delta^m), y^m)\}$.

- A pathwise approach (**linSES**):

  (1) Calculate the pathwise signature of each path $X^{i,j}$ truncated at a fixed level l_1, i.e, for any given partition $0 = t_0 < t_1 < \cdots < t_n = T$, calculate $S^{l_1}(X_{[0,t_i]}), \forall t_i$.

(2) Average the pathwise signature of paths from the same group to get the expected pathwise signature which has dimension $n \times d$ with $d = 2^{l_1+1} - 1$.

(3) For the new expected pathwise signature, calculate its signature truncated at level l_2, then we get $\frac{d^{l_2+1}-1}{d-1}$ features.

(4) Apply the Lasso regression method to select features from the new $\frac{d^{l_2+1}-1}{d-1}$ feature and fit the linear regression model.

For more details and mathematical backgrounds for these two models, please refer to [91].

2.3.4 RNN models

Recurrent neural networks (RNNs [48]) have been shown to be effective in machine learning applications based on time series, such as natural language processing (NLP). Given a sequential input $(x_1, \cdots, x_n)$, where $x_k \in \mathbb{R}^d, \forall k \in \{1, \cdots, n\}$ and d is the dimension of the observation, at each time step t, an RNN generates a hidden state h_t based on the current input x_t and the previous hidden state h_{t-1} by:

$$h_t = f(x_t, h_{t-1}) \tag{2.23}$$

Additionally, at each time step, an optional output can be generated by $y_t = g(h_t)$.

However, classical RNN models struggle to learn longer-term dependencies and have issues with vanishing or exploding gradients during backpropagation through time [92]. Long Short-Term Memory (LSTM) [49] and its variant, Gated Recurrent Units (GRU) [53], were introduced to address these problems. LSTMs and GRUs are designed with gating mechanisms that regulate the flow of information and enable better capturing of long-range dependencies. These architectures have been widely adopted and have shown

improved performance in various sequence modeling tasks.

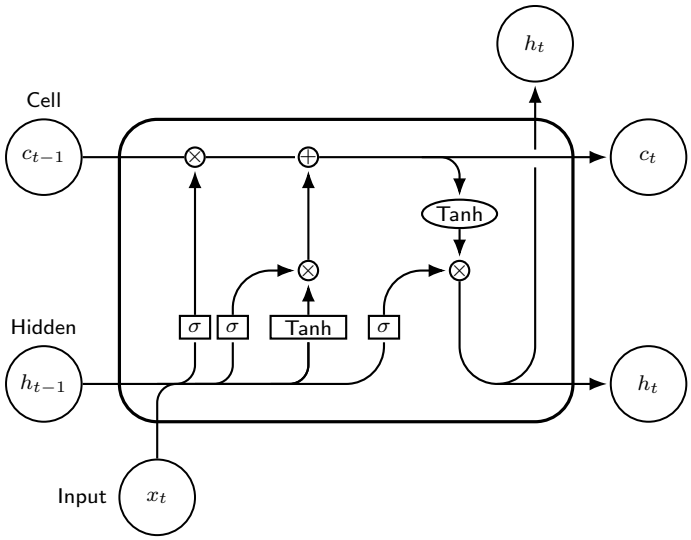


Figure 2.4: An illustration of the LSTM model architecture at time t.

Figure 2.4 illustrates the architecture of the Long Short-Term Memory (LSTM) model at each time step t. The sigmoid function is denoted as $\sigma(x) = \frac{1}{1+e^{-x}}$. At each step t, the LSTM maintains an input gate i_t, a forget gate f_t, an output gate o_t, and a cell state c_t to store long-term memory. These components are updated as follows:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i), \tag{2.24}$$

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f), \tag{2.25}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o), \tag{2.26}$$

$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c), \tag{2.27}$$

$$c_t = i_t \odot \tilde{c}_t + f_t \odot c_{t-1}, \tag{2.28}$$

$$h_t = o_t \odot \tanh(c_t), \tag{2.29}$$

where W_*, U_* are the weight matrices and b_* are the bias vectors. The operation $\odot$

denotes the element-wise vector product.

Gated Recurrent Units (GRUs) [53] are a lightweight variant of LSTMs, sharing a similar architecture but requiring fewer parameters to train. GRUs have shown comparable performance to LSTMs in various sequence modeling tasks while offering computational efficiency.

At each time step t, a GRU updates a reset gate r_t and an update gate z_t as follows:

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r), \tag{2.30}$$

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z), \tag{2.31}$$

$$\tilde{h}_t = \tanh(W x_t + U(r_t \odot h_{t-1}) + b), \tag{2.32}$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t. \tag{2.33}$$

The reset gate r_t determines how much of the previous hidden state should be retained, while the update gate z_t controls the balance between the previous hidden state and the candidate hidden state $\tilde{h}_t$. This gating mechanism allows GRUs to effectively capture long-term dependencies while requiring fewer parameters compared to LSTMs.

2.4 Experiments and Results

2.4.1 Classification

In our analysis, we focus on the $ARMA(1,1)$ model and generate simulations for four distinct classes of time series, each with a length of 100 observations. For every class, we simulate 500 paths, incorporating white noise represented as $\varepsilon_t \sim N(0,1)$. The

generation process for each time series class is governed by the equations below:

$$\begin{aligned}
\text{Class 0:} \quad & Y_t = 0.4Y_{t-1} + \varepsilon_t + 0.5\varepsilon_{t-1} \\
\text{Class 1:} \quad & Y_t = 0.8Y_{t-1} + \varepsilon_t + 0.7\varepsilon_{t-1} \\
\text{Class 2:} \quad & Y_t = -0.4Y_{t-1} + \varepsilon_t + 0.5\varepsilon_{t-1} \\
\text{Class 3:} \quad & Y_t = -0.8Y_{t-1} + \varepsilon_t + 0.7\varepsilon_{t-1}
\end{aligned}$$

Figure 2.5 presents a comparison between the different time series. The left panel compares time series from Class 0 and Class 1, which have positive autoregressive (AR) coefficients, while the right panel compares time series from Class 2 and Class 3, with negative AR coefficients.

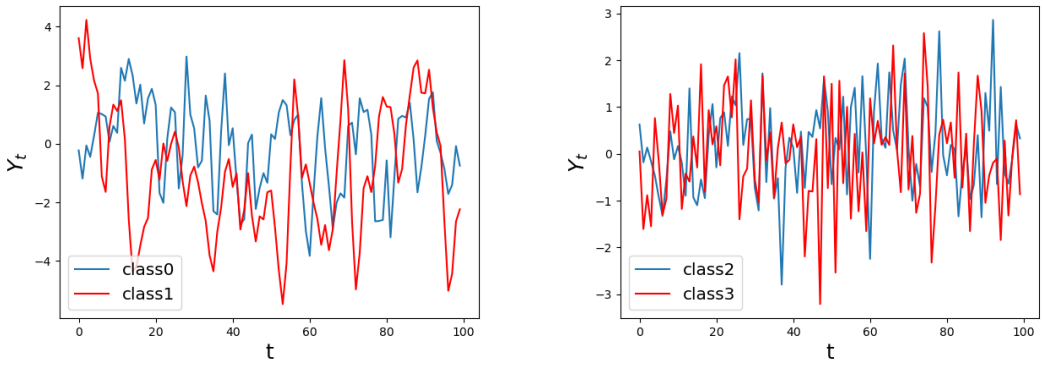
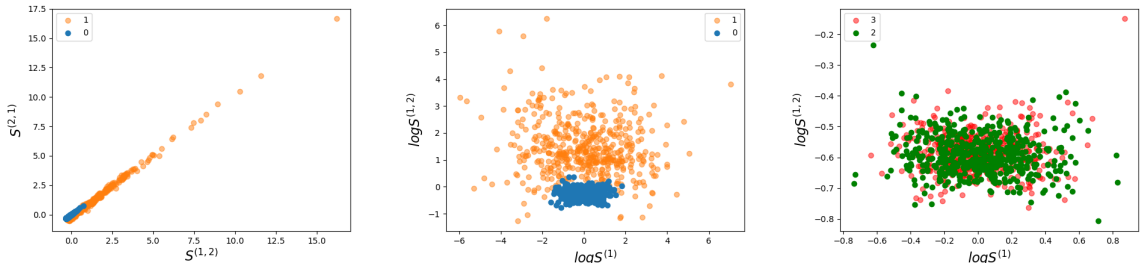


Figure 2.5: Four different time series. (Left) Comparison between time series from Class 0 and Class 1 with positive AR coefficients. (Right) Comparison between time series from Class 2 and Class 3 with negative AR coefficients.

We can observe from Figure 2.5 that the time series in the right panel, with negative AR coefficients, exhibit more drastic changes compared to those in the left panel with positive AR coefficients. This difference in behavior can be attributed to the sign of the AR coefficients, which significantly influences the behavior and evolution of the time series.

For any given path $Y = (Y_1, \cdots, Y_n)$, we employ the lead-lag transformation (see Equation 2.2.4) to extend the original path into a two-dimensional form, upon which we then apply the signature transformation.



(a) Signature comparison (b) Logsignature comparison, Class 0 and 1 (c) Logsignature comparison, Class 2 and 3

Figure 2.6: (Left) Comparison between two signature values of paths from Class 0 and Class 1. (Middle) Comparison between two log-signature values of paths from Class 0 and Class 1. (Right) Comparison between two log-signature values of paths from Class 2 and Class 3.

We can observe from Figure 2.6 that the left panel shows the scatter plot of $S(P_0)^{(1,2)}$ versus $S(P_1)^{(2,1)}$, where P_0 and P_1 are the paths simulated from Class 0 and Class 1, respectively. From the left panel, we note that the two elements in the signature vector are more widely distributed for Class 1, but it is difficult to distinguish the classes solely based on these two elements. In the middle panel, we can see that the scatter plot of $\log(S(P_0)^{(1)})$ versus $\log(S(P_1)^{(1,2)})$ exhibits a clear separation between the paths from these two classes. This observation indicates that the logsignature can help distinguish time series effectively, similar to the signature. However, the logsignature has a shorter length compared to the signature because it retains most of the essential information from the original time series while removing redundant information present in the signature vector.

For the right panel, we observe that even the logsignature fails to distinguish paths

from Class 2 and Class 3. This might be due to the fact that paths from these two classes are negatively related and therefore exhibit more frequent changes within the time interval, making it challenging for the signature and logsignature vectors to extract useful information for classification.

Subsequently, we partition the observations from Classes 0 and 1 into training and test sets in a 7:3 ratio. We apply both the signature and log-signature transformations, followed by logistic regression with an l_1 penalty for feature selection. The results are presented in the confusion matrices below:

Table 2.1: Signature

TruePredicted	0	1
0	144	2
1	9	145

Table 2.2: logSignature

TruePredicted	0	1
0	142	4
1	14	140

We can observe that both the signature (Table 2.1) and logsignature (Table 2.2) methods perform well in classifying observations from Class 0 and Class 1. The confusion matrices show that the models are able to accurately distinguish between the two classes, with high true positive rates and low false positive rates.

2.4.2 Missing data

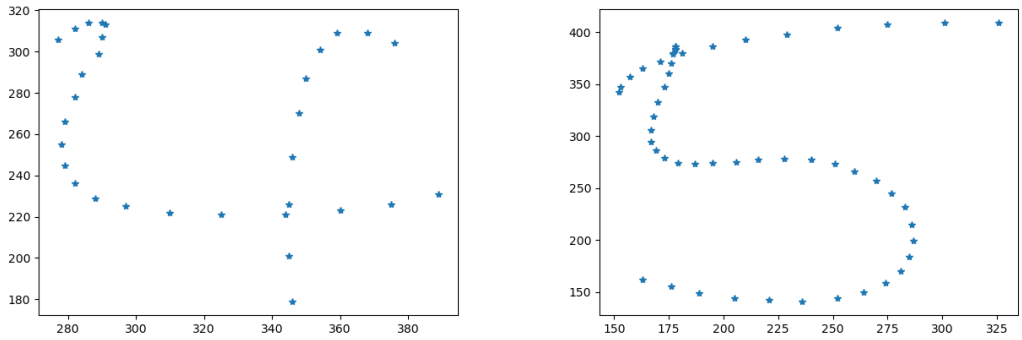


Figure 2.7: Examples of hand-written numbers from the pen-digit dataset. (Left) Path of the number 4. (Right) Path of the number 5.

In this section, we use the pen-digit dataset. For the raw data and the code for data cleaning, please refer to Pen-digit Data Clean. From Figure 2.7, we can observe that there are obvious gaps between consecutive points in some of the handwritten numbers. Although these gaps do not represent missing values, they arise from the discontinuity in the writing process. To address this issue, we treat these gaps as missing values and augment the raw data with an "ink" dimension, which records the amount of int spent while writing the number.

Figure 2.8 shows the distribution of the maximum distances between consecutive points from records in the Pen-digit dataset. From the picture, we can observe that most handwritten numbers do not have gaps greater than 30. Therefore, we set a threshold of 30 to identify gaps between consecutive points. Based on this observation, we add the "ink" dimension following the rule: when $d(X_{n-1}, X_n) > 30$, $\text{ink}_n = \text{ink}_{n-1}$, which means we did not spend any ink between the two points. When $d(X_{n-1}, X_n) < 30$, $\text{ink}_n = \text{ink}_{n-1} + 0.01$, indicating that we spent ink while writing down these consecutive points.

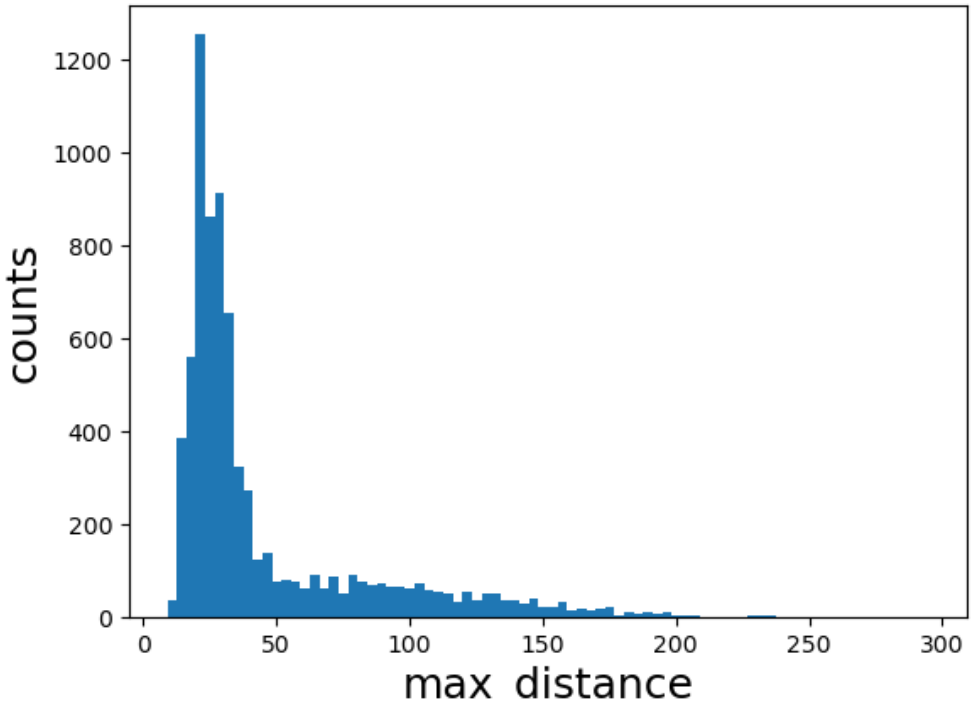


Figure 2.8: Histogram of maximum distances between two continuous points from the Pen-digit dataset.

After the augmentation, we apply the Min-Max transformation

$$\tilde{x} = 2 \cdot \frac{x - \min(x)}{\max(x) - \min(x)} - 1$$

to the xy-coordinates, restricting all points to the square $[-1, 1] \times [-1, 1]$. Then, we apply the signature and logsignature transformations truncated at level 4 separately to the raw data and augmented data and fit a linear regression model. We compare the true classes with the predicted classes over the test dataset, and the precision rates are presented in the following table:

From Table 2.3, we can observe that after using the missing data augmentation, the precision rate is significantly improved. The logSignature method uses fewer features

Table 2.3: Prediction precision rate

	Signature	logSignature
Raw-data	0.877	0.831
Augmented-data	0.942	0.902

than the signature method but achieves better performance.

Next, with the idea from [93], we will train the GRU model based on the raw data and signatures. Since in each batch, the GRU model requires the input sequences to have equal length, we will append $[0, 0]$ to short paths and randomly drop some points from long paths to make them have equal length of 50. Therefore, the input has a shape of 50×2. To use the signature transformation, I divided the new paths into 3 parts and calculated the signatures truncated at level 4 of each part. Besides, due to the shift-invariance of the signature, I also added the first observation of each subpath to the signature vector, which indicates the position of the subpath in the square. Consequently, the Sig-GRU model has an input shape of 3×32. Additionally, we tried training the GRU model based on the log signatures of the augmented data, which has an input shape of 3×34.

Table 2.4: GRU Model Comparison

	Training Time (s)	Prediction Precision
GRU	160.49	95.62
Sig-GRU	15.04	91.97
logSig-GRU	15.21	93.20

From Table 2.4, we can observe that the GRU models indeed improve the prediction precision rate compared to the classical logistic regression model. Additionally, we can see that the GRU model has the greatest prediction precision, but the training time is quite long. However, the signature-GRU and logsignature-GRU models have similar prediction performance but require much shorter training times.

2.4.3 Parameter Estimation

In this section, we will apply the two methods mentioned in Section 2.3.3: **kerES** and **linSES** to see how signature transformation performs in estimating parameters.

The *fractional Ornstein-Uhlenbeck* (fOU) process is expressed as:

$$dP_t = -a(P_t - m)dt + \gamma dW_t^H,$$

where $a > 0$ is the parameter we want to estimate, P_t is the stock price, $m = 0.5$ is the mean of the price, $\gamma = 0.08$ is the volatility coefficient, and W_t^H is the fractional Brownian Motion (fBM) with Hurst parameter H. The volatility process is defined as $\sigma_t = \exp(P_t)$.

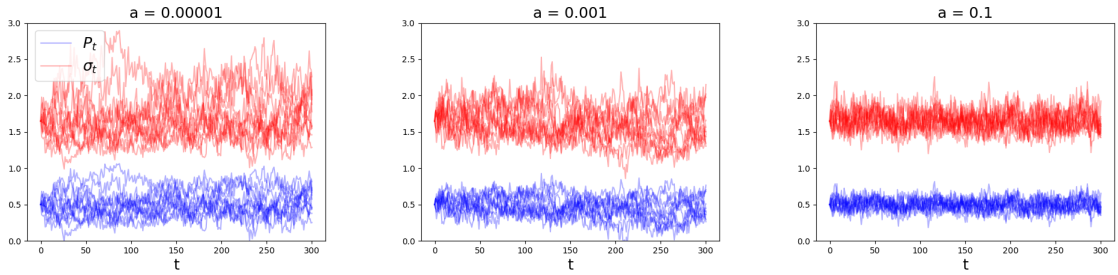
The fractional Brownian Motion W_t^H is a continuous-time Gaussian process that satisfies the following properties:

$$\mathbb{E}[W_t^H] = 0,$$
$$\mathbb{E}[(W_t^H - W_s^H)^2] = \sigma^2 |t - s|^{2H},$$

where σ^2 is the variance parameter, and $H \in (0, 1)$ is the Hurst parameter. The Hurst parameter H controls the roughness or smoothness of the fractional Brownian Motion. When $H = 0.5$, fBM reduces to standard Brownian Motion with independent increments. When $H > 0.5$, the increments exhibit positive correlation or persistence, resulting in a smoother behavior. Conversely, when $H < 0.5$, the increments exhibit negative correlation or anti-persistence, leading to a rougher behavior.

In our experiment, we set $H = 0.2$, which gives us rough paths.

From Figure 2.9, we can observe that different values of a have an influence on the variance of the fOU process paths when other parameters are fixed. The larger the value



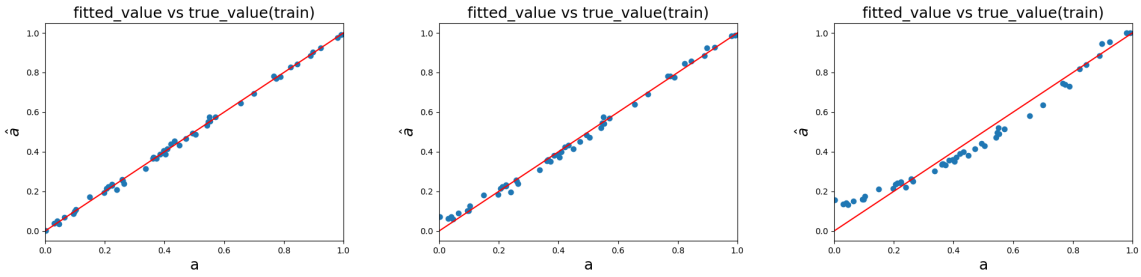
(a) a = 0.00001 (b) a = 0.001 (c) a = 0.1

Figure 2.9: Examples of fOU paths driven by different **a** values.

of a, the less volatile the entire path will be. We will randomly draw $\{a_j\}_{j=1}^{50}$, where $a_j \sim \text{Unif}(0,1)$. For each selected a_j, we simulate 50 fOU volatility paths $\{\sigma_t^{i,j}\}_{i=1}^{50}$ driven by a_j and use **kerES** and **linSES** for parameter estimation.

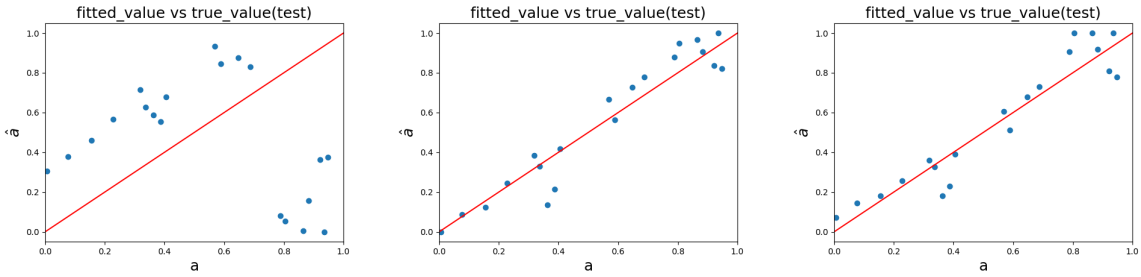
In **kerES**, we calculate the expected signature truncated at level 5. For the **linSES** method, since the length of the simulated path is 300 ($0 = t_0 < t_1 < \cdots < t_{300} = 1$), it requires a lot of time and storage space to find the whole pathwise signature. To save time and storage, we calculate the signature truncated at level 3 for the path $\sigma_{[0,30k]}$ for $k \in \{1, 2, \cdots, 10\}$. Then, for the new 10×14 path, we calculate the signature truncated at level 2. Inspired by the fact that the logSignature transformation uses fewer features and achieves almost the same performance, we also try to calculate the pathwise logSignature truncated at level 4 in **linSES**, and for the new 10×8 matrix, we calculate the logsignature truncated at level 3.

Figures 2.10 and 2.11 depict the prediction performance of these two methods, as well as the **log-signature linSES**, on the training set and test set, respectively. From the figures, we can observe that all methods work, and **linSES** and **log-signature linSES** perform better on the test set. However, this improved performance comes at the cost of more training time and storage space. Additionally, the **log-signature linSES** does not outperform the **linSES**.



(a) kerES (b) linSES-Signature (c) linSES-logSignature

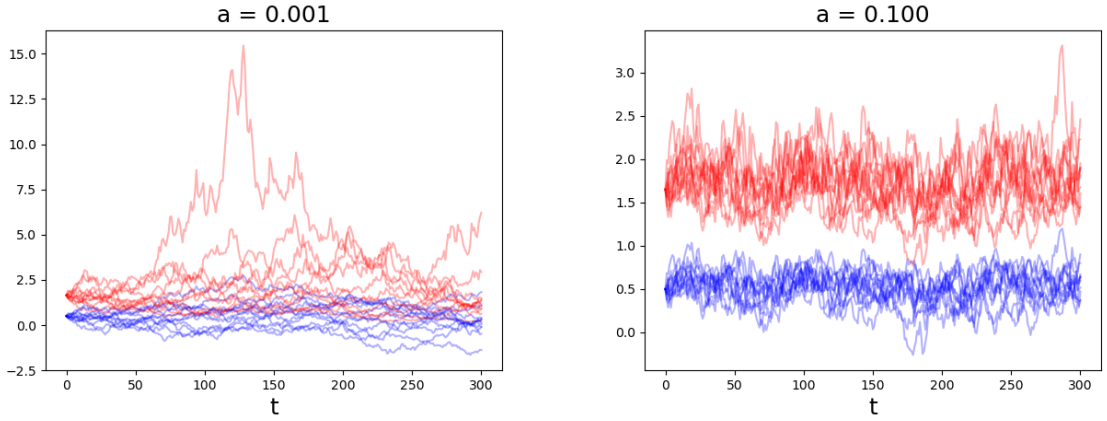
Figure 2.10: Performance of kerES and linSES on the training set.



(a) kerES (b) linSES-Signature (c) linSES-logSignature

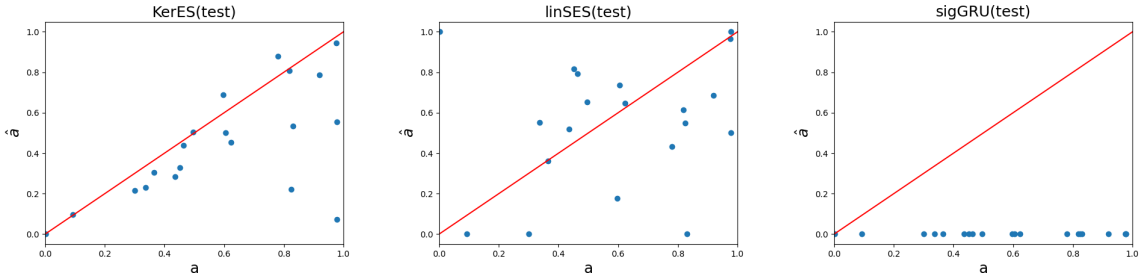
Figure 2.11: Performance of kerES and linSES on the test set.

Next, we compare the **kerES**, **linSES**, and **sigGRU** models under the condition $H = 0.5$. In this case, the model is the classical Ornstein-Uhlenbeck (OU) process.



(a) a = 0.001 (b) a = 0.1

Figure 2.12: Examples of OU paths driven by different **a** values.



(a) kerES (b) linSES (c) SigGRU

Figure 2.13: Performance of kerES, linSES and SigGRU on the test set.

Figure 2.13 shows a different result from the previous one. In this example, the **kerES** model performs better than the **linSES** model, while the **SigGRU** seems not to work at all.

2.4.4 Frequency Detection

Seasonality, a recurring pattern within time series data, significantly influences various scientific and economic domains. Unraveling these periodic trends is paramount for accurate forecasting, anomaly detection, and strategic decision-making. The two most widely used techniques for seasonality or frequency detection are Fast Fourier Transform (FFT) and Lomb-Scargle analysis, particularly adept at handling regular and irregular time series, respectively.

1. **Fast Fourier Transform:** The FFT algorithm ([94]) efficiently computes the Discrete Fourier Transform (DFT), which transforms a time series from the time domain into the frequency domain. In the DFT formula:

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-\frac{i2\pi}{N} kn} \tag{2.34}$$

where

- $X(k)$ is the amplitude and phase of the frequency component at frequency k;
- $x(n)$ is the observed time series with a total of N observations.

When k corresponds to a true frequency in the time series, $X(k)$ exhibits a peak, indicating a strong presence of that frequency in the data. Thus, the magnitudes $|X(k)|$ for dominant frequencies will be significantly higher than others in the time series. Therefore, the FFT allows the identification of dominant frequencies that represent the underlying seasonalities.

2. **Lomb-Scargle Periodogram:** While FFT is proficient with evenly spaced data, the Lomb-Scargle analysis ([95], [96]) extends the exploration of seasonality into the realm of irregular time series. Its formula:

$$P(\omega) = \frac{1}{2\sigma^2} \left\{ \frac{\left[\sum_{n=0}^{N-1}(x_n - \bar{x})\cos\omega(t_n - \tau)\right]^2}{\sum_{n=0}^{N-1}\cos^2\omega(t_n - \tau)} + \frac{\left[\sum_{n=0}^{N-1}(x_n - \bar{x})\sin\omega(t_n - \tau)\right]^2}{\sum_{n=0}^{N-1}\sin^2\omega(t_n - \tau)} \right\} \tag{2.35}$$

where

- σ^2 is the variance of the observed data;
- τ is a time offset that makes the periodogram invariant to shifts in time;
- $P(\omega)$ represents the power associated with a particular frequency ω.

A significant peak in $P(\omega)$ at a certain frequency indicates a strong periodic component at that frequency.

The Lomb-Scargle method is particularly salient in astronomical observations, environmental studies, and any field grappling with data irregularities, as it maintains high sensitivity to periodic signals even with missing or sparse data points.

In our study, we also try to apply the signature transformation to detect frequencies of time series. Here are the steps of our method:

1. Apply the Lead-Lag transformation (2.2.4) if the original time series is 1-dimensional, so that we can calculate its signature vectors.

2. For all possible frequencies $f = \frac{1}{d}$, where d is the duration of the seasonality ranging from 1 to $\frac{l}{2}$, and l is the length of the entire time series, we calculate the signature vectors of all subpaths with length d, and then compute the average l_2 distance between each vector.

3. Sort all average l_2 distances, and the smallest k distances represent the k most likely frequencies.

The proposed algorithm, while theoretically sound, failed to perform well in detecting frequencies accurately on real-world datasets. Despite our initial motivation that signature vectors could potentially serve as an alternative to Fourier-based methods for frequency detection, especially for irregular time series, the experimental results were unsatisfactory. Several factors may contribute to the algorithm's suboptimal performance:

1. **Truncation of Signature Vectors**: In practice, we truncate the signature (or log-signature) vectors at a certain level N for computational feasibility. However, this truncation can lead to information loss, as the infinite signature vector is a one-to-one mapping of the original time series. Truncating the signature vector at level N may miss some higher-order features essential for accurate frequency detection.

2. **Magnitude Variations in Higher-Order Signatures**: The signature terms are computed as iterated integrals, and their magnitudes can either explode or

diminish for higher levels, depending on the characteristics of the time series. The magnitude variations in higher-order signature terms can cause the l_2 distance, used as a similarity measure between signature vectors, to be dominated by a few large terms or insensitive to small but informative terms, hindering accurate frequency detection.

3. **Non-stationarity and Noise**: Real-world time series data often exhibits non-stationarity and is contaminated with noise and irregularities. These factors can distort the signature representation and adversely affect the frequency detection process.

4. **Choice of Lead-Lag Transformation**: The Lead-Lag transformation used to generate higher-dimensional paths from a one-dimensional time series may not be optimal for frequency detection, and alternative transformations or pre-processing steps could be more suitable.

Despite our initial motivation to leverage signature vectors for frequency detection, particularly for irregular time series, the experimental results indicate that this approach cannot reliably match the accuracy of well-established methods like the Fast Fourier Transform (FFT). Further research is needed to explore alternative techniques, similarity measures, or a combination of signature transformations with other frequency analysis methods to improve the performance of signature-based approaches for frequency detection.

Chapter 3

Neural Differential Equations

3.1 Neural Ordinary Differential Equations (NODEs)

3.1.1 Overview

Introduction to Neural ODEs

Neural Ordinary Differential Equations (Neural ODEs), as introduced by [61], represent a revolutionary approach in the field of deep learning, blending the discrete computational models traditionally used in machine learning with the continuous models prevalent in physical sciences and engineering. Unlike conventional neural networks, such as feedforward and recurrent neural networks that operate on discrete data across spatial or temporal dimensions, Neural ODEs are designed to model continuous-time dynamics. This distinction allows Neural ODEs to encapsulate the inherently continuous nature of many real-world phenomena, evolving over time in a way that traditional models may not accurately capture.

At the core of Neural ODEs is the concept of parameterizing the derivative of a hidden state with respect to a continuous variable, such as time, through a neural network. This

neural network learns the vector field dictating the hidden state's evolution across the input range, effectively bridging the gap between discrete layer-based architectures and continuous-depth models. The inspiration for Neural ODEs can be traced back to the observation that Residual Networks (ResNets) [97] can be viewed as an Euler discretization of continuous transformations, as governed by Ordinary Differential Equations (ODEs).

This continuous-depth approach to modeling allows Neural ODEs to achieve several significant advantages over traditional architectures. It enables constant-memory computation by calculating the hidden state on-the-fly during evaluation, without the need to store intermediate values. Additionally, Neural ODEs offer flexibility in handling irregularly sampled data and provide a natural framework for interpreting continuous-time processes. The integration of ODE solvers at inference time allows for the network's evolution to be modeled as a continuous process, enhancing the representational power and interpretability of Neural ODEs in domains where continuous dynamics are crucial.

Problems Addressed

Neural ODEs elegantly address several challenges inherent to discrete models:

- **Memory Efficiency**: By leveraging the adjoint method for backpropagation, Neural ODEs significantly reduce the memory footprint during training, as they do not require storing intermediate states.
- **Handling of Irregular Time Series**: Their continuous nature makes Neural ODEs particularly suited for modeling time series data sampled at irregular intervals, a common scenario in many real-world applications.
- **Flexibility and Generalization**: Neural ODEs can theoretically represent an infinite depth network, providing a flexible framework that can adapt to the complexity of the data.

Important Follow-up Work

Following the foundational work on Neural ODEs, several significant research directions have emerged, focusing on extending the framework and enhancing its practicality and efficiency. Key developments include:

- **Augmented Neural ODEs** enhance the model's ability to learn complex functions by augmenting the state space, thus improving stability and generalization [63].
- **Neural Stochastic Differential Equations (SDEs)** extend Neural ODEs to stochastic settings, allowing for the modeling of systems influenced by inherent randomness, thereby broadening the scope of applications significantly [68, 69, 70].
- **Neural Controlled Differential Equations (CDEs)** offer a framework for efficiently handling irregular time series by incorporating control signals into the continuous dynamics, enhancing their applicability to a wide range of real-world datasets [72].
- **FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models** utilizes continuous dynamics for reversible generative modeling, allowing unrestricted neural network architectures in the dynamics function for improved density estimation and variational inference [62].
- **Neural Manifold ODEs** aim to model data residing on manifolds with continuous dynamics, facilitating the processing of complex geometrical and topological data structures [65].
- Some works focus on accelerating the training process of Neural ODEs, addressing the challenges of training stability and efficiency: [66] delves into Jacobian and

kinetic regularization, while [67] optimizes the adjoint method for quicker and more memory-efficient backpropagation.

This vibrant trajectory of research reflects the deepening and broadening of Neural ODEs' theoretical foundations and practical applications, illustrating the community's ongoing commitment to refining and expanding this versatile modeling framework.

3.1.2 Mathematical Backgrounds

ODE Review

An Ordinary Differential Equation (ODE) describes the relationship between a function and its derivatives, providing a way to model the change in a system over a continuous variable, typically time. Formally, an ODE for a function $y(t)$ is given by:

$$\frac{dy(t)}{dt} = f(t, y(t)), \tag{3.1}$$

where $f(t, y(t))$ is a function of t and $y(t)$. The solution to an ODE is the function $y(t)$ that satisfies this equation for given initial conditions $y(t_0) = y_0$.

Classical solvers for ODEs range from simple methods like the Euler approximation to more complex, adaptive step-size methods. The Euler method, in particular, updates the value of $y(t)$ in discrete steps as follows:

$$y_{n+1} = y_n + hf(t_n, y_n), \tag{3.2}$$

where h is the step size. This discrete update mechanism is conceptually similar to the way Residual Networks (ResNets) update their hidden states:

$$h_{n+1} = h_n + f(h_n, \theta_n), \tag{3.3}$$

with θ_n representing the parameters of the n-th layer of the network, and $f(\cdot, \cdot)$ embodying the layer's transformation function. This resemblance between the discrete steps of Euler's method for solving ODEs and the layer-wise updates in ResNets inspired the development of Neural ODEs. By modeling the hidden state transformation as a continuous process governed by an ODE, Neural ODEs extend the discrete update strategy of ResNets into a continuous domain, allowing for an infinitely deep model conceptualized through continuous dynamics.

Theorem 5. (Picard–Lindelöf Theorem [98]) Let $D \subseteq \mathbb{R} \times \mathbb{R}^n$ be a closed rectangle with $(t_0, y_0) \in D$. Let $f : D \to \mathbb{R}^n$ be a function that is continuous in t and Lipschitz continuous in y. Then, there exists some $\varepsilon > 0$ such that the initial value problem:

$$y'(t) = f(t, y(t)), \quad y(t_0) = y_0. \tag{3.4}$$

has a unique solution $y(t)$ on the interval $[t_0 - \varepsilon, t_0 + \varepsilon]$.

This theorem is crucial in the analysis of ODEs as it guarantees the existence and uniqueness of solutions under certain conditions, a property that will be later leveraged to discuss the efficacy of the Neural ODE model.

NODE Algorithm

Neural Ordinary Differential Equation (NODE) Algorithm The Neural ODE algorithm leverages the continuous nature of ODEs to model the transformation of hidden states in a neural network. This continuous transformation is parameterized by a neural network, termed as the dynamics function $f_\theta(t, h(t))$, where $h(t)$ represents the hidden state at time t, and θ denotes the parameters of the neural network. The evolution of

$h(t)$ from an initial state $h(t_0)$ to a final state $h(t_1)$ is governed by the following ODE:

$$\frac{dh(t)}{dt} = f_\theta(t, h(t)). \tag{3.5}$$

Adjoint Method The key to efficiently training Neural ODEs lies in the adjoint method, a technique for calculating gradients of a cost function L with respect to the parameters θ, without retaining the entire trajectory of $h(t)$ during backpropagation. This method computes the gradient $\frac{dL}{d\theta}$ by solving an additional ODE backward in time.

The adjoint state $a(t) = \frac{\partial L}{\partial h(t)}$ captures the sensitivity of the loss with respect to the hidden state at any time t. The evolution of $a(t)$ is described by the adjoint equation:

$$\frac{da(t)}{dt} = -a(t)^T \frac{\partial f_\theta(t, h(t))}{\partial h(t)}. \tag{3.6}$$

Solving this equation backward in time, from t_1 to t_0, allows us to compute the gradients with respect to the hidden states throughout the trajectory.

To update the parameters θ, we also need to compute the gradient of the loss with respect to θ. This is achieved by integrating another term alongside the adjoint state:

$$\frac{dL}{d\theta} = -\int_{t_1}^{t_0} a(t)^T \frac{\partial f_\theta(t, h(t))}{\partial \theta} dt. \tag{3.7}$$

By solving this integral backward in time, the adjoint method efficiently computes the gradient of the loss with respect to the parameters θ, enabling the update of the model parameters using gradient descent or other optimization algorithms.

This adjoint-based approach significantly reduces the computational cost and memory requirements of training Neural ODEs, as it obviates the need to store the forward pass's intermediate states. Moreover, it aligns with the continuous nature of Neural ODEs, allowing for the flexible adjustment of the model's complexity by simply changing the

integration bounds.

NODE for Time Series Prediction

Loss Functions in Time Series Prediction To guide the learning process, a loss function is employed to quantify the discrepancy between the model's predictions and the actual data. The choice of the loss function can vary based on the project's objectives, with common options including Mean Squared Error (MSE), Mean Absolute Error (MAE), and Symmetric Mean Absolute Percentage Error (SMAPE).

- **Mean Squared Error (MSE)**: The MSE is widely used for regression tasks, including time series prediction. It calculates the average of the squares of the errors between the predicted and actual values. Mathematically, it is defined as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{3.8}$$

  where y_i represents the actual value and $\hat{y}_i$ the predicted value for the i-th data point, and n is the number of data points.

- **Mean Absolute Error (MAE)**: The MAE measures the average magnitude of errors in a set of predictions, without considering their direction. It is defined as:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|, \tag{3.9}$$

  providing a straightforward measure of prediction accuracy with the same units as the data.

- **Symmetric Mean Absolute Percentage Error (SMAPE)**: SMAPE is an accuracy measure based on percentage errors, which is less sensitive to large outliers

than MSE and MAE. SMAPE is defined as:

$$\text{SMAPE} = \frac{100\%}{n} \sum_{i=1}^{n} \frac{|y_i - \hat{y}_i|}{(|y_i| + |\hat{y}_i|)/2}, \tag{3.10}$$

making it a useful metric when dealing with datasets that span several orders of magnitude and where the relative error is more meaningful.

These metrics are instrumental in evaluating the efficacy of predictive models by quantitatively measuring the discrepancies between the predicted outputs and the actual observed values. This comparison not only serves as a benchmark for model performance but also directs the adjustment of the model's parameters to improve its performance.

NODE-VAE Architecture A Variational Autoencoder (VAE) is a generative model that leverages deep learning and Bayesian inference to learn latent representations of high-dimensional data. The model consists of two main components: an encoder and a decoder. The encoder compresses the data into a latent space representation, while the decoder reconstructs the data from this latent representation. The objective of a VAE is to maximize the Evidence Lower Bound (ELBO), balancing reconstruction accuracy with model complexity.

For time series data $\mathbf{X} \in \mathbb{R}^{d \times n}$, with d denoting the feature dimension and n the number of timestamps, traditional VAEs face challenges due to the discrete and potentially irregular temporal nature of the observations. Here, $\mathbf{X}$ comprises sequences of feature vectors at timestamps $\{\mathbf{X}_{t_0}, \ldots, \mathbf{X}_{t_n}\}$, where the interval $t_j - t_{j-1}$ may vary.

Integrating Neural Ordinary Differential Equations (NODEs) within the VAE framework provides a solution to these challenges by modeling the latent variables' continuous dynamics. This approach is particularly suited for handling irregular sampling intervals in time series data.

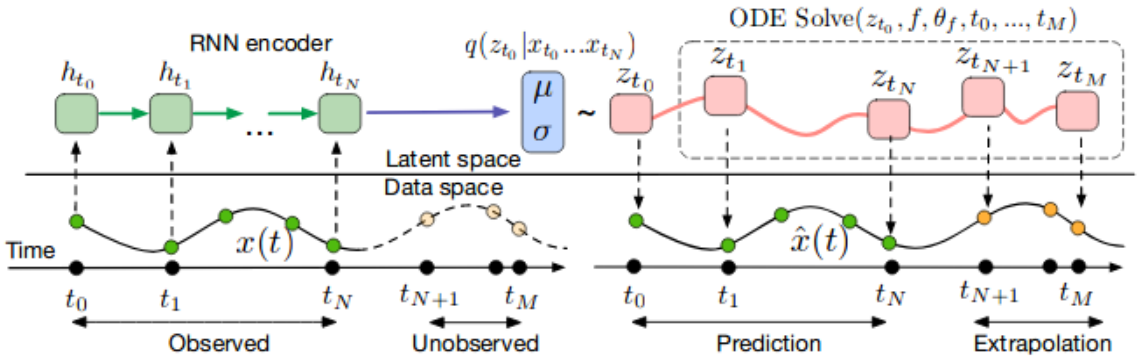


Figure 3.1: NODE-VAE architecture for time series prediction

From figure 3.1 [61], we can see that given a time series $\mathbf{x}(t) = \{\mathbf{x}_{t_0}, \mathbf{x}_{t_1}, \ldots, \mathbf{x}_{t_N}\}$, the RNN encoder processes the data sequentially to produce hidden states $\mathbf{h}_t$ that capture the temporal dependencies. Then, the latent distribution at the initial time point t_0 is parameterized by the mean μ and variance σ^2 derived from the RNN encoder:

$$q(\mathbf{z}_{t_0}|\mathbf{x}_{t_0}, \ldots, \mathbf{x}_{t_N}) = \mathcal{N}(\mu, \sigma^2). \tag{3.11}$$

A sample $\mathbf{z}_{t_0}$ from this distribution initializes the NODE, which is defined by the differential equation:

$$\frac{d\mathbf{z}(t)}{dt} = f(\mathbf{z}(t), t; \theta). \tag{3.12}$$

Here, f is a neural network parameterized by θ, encoding the time derivatives of the latent states. This NODE is capable of evolving the latent states across arbitrary time points, enabling interpolation and extrapolation to predict future states $\mathbf{z}_{t_m}$, which the decoder then maps to the predicted time series $\hat{\mathbf{x}}(t)$.

The training of the NODE-VAE model is driven by the maximization of the Evidence

Lower Bound (ELBO), given by:

$$\mathcal{L}(\theta; \mathbf{x}) = \mathbb{E}_{q(\mathbf{z}_{t_0}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z}_{t_0})] - \mathrm{KL}[q(\mathbf{z}_{t_0}|\mathbf{x})||p(\mathbf{z}_{t_0})]. \tag{3.13}$$

The ELBO is composed of two key terms: The first term $\mathbb{E}_{q(\mathbf{z}_{t_0}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z}_{t_0})]$ is the expected log likelihood of the observed data, which reflects the accuracy of data reconstruction from the latent variables. This term encourages the decoder to generate data closely resembling the original input. Maximizing this term directly reduces the reconstruction error, compelling the latent variables to capture the essential features of the data for accurate reconstruction.

The second term $\mathrm{KL}[q(\mathbf{z}_{t_0}|\mathbf{x})||p(\mathbf{z}_{t_0})]$ is the KL divergence, acting as a regularizer that measures how much the approximate posterior distribution of the latent variables deviates from their prior distribution. Minimizing the KL divergence enforces the encoded latent variables to remain close to the prior, encouraging the latent space to be well-structured and avoiding overfitting to the training data.

By maximizing the ELBO, we aim to train a model that not only reconstructs the training data accurately but also captures the underlying probabilistic structure, ensuring that the model can generalize well and generate new, plausible time series sequences.

[64] explores various configurations within the VAE framework to effectively model time series data that exhibit irregular sampling. The paper investigates different combinations of encoders and decoders, specifically contrasting ODE-based and RNN-based architectures. The configurations examined include ODE-ODE, where both the encoder and decoder are ODEs; RNN-ODE, with an RNN encoder and an ODE decoder; and RNN-RNN, where both components are RNNs.

Empirical results from the study reveal that the ODE-ODE configuration yields the best performance among the architectures tested. This configuration, by employing con-

tinuous dynamics models for both encoding and decoding phases, demonstrates a superior ability to handle the challenges posed by irregular sampling in time series data. The continuous nature of ODEs offers a natural and coherent approach for modeling time series data, providing flexibility and robustness that are advantageous for capturing complex temporal patterns, especially when the time points are non-uniformly distributed.

3.1.3 Important follow-up works

Following the introduction of Neural Ordinary Differential Equations (NODEs), researchers have explored various directions to extend their applicability and address inherent limitations. In this section, we will focus on two significant developments: Augmented Neural ODEs (ANODEs) and Neural Manifold ODEs.

To better understand the theoretical foundations and practical advantages of these advancements, we first need to review the concept of manifolds, which plays a central role in these extensions. The notion of manifolds, along with the associated tangent spaces and geometric properties, will provide the necessary mathematical background to understand how ANODEs and Neural Manifold ODEs improve the traditional NODE framework by accommodating more complex geometrical and topological structures.

Other dynamic function approximation methods, such as Neural Stochastic Differential Equations (NSDEs), which also present important extensions of NODEs, will be discussed in detail in the subsequent section.

Manifold Review

A manifold is a mathematical space that, at a small enough scale, resembles the Euclidean space and can be mapped to coordinate planes or open subsets of Euclidean space. Formally, a manifold $\mathcal{M}$ can be defined as a set that is a Hausdorff space, with a count-

able basis, and locally homeomorphic to $\mathbb{R}^n$, making it a locally Euclidean topological space of dimension n.

Definition 11 (Manifold)**.** A **manifold** of dimension n, denoted as $\mathcal{M}$, is a topological space where every point $p \in \mathcal{M}$ has a neighborhood U that is homeomorphic to the Euclidean space $\mathbb{R}^n$. This homeomorphism $\phi : U \to \mathbb{R}^n$ is called a **chart**. The dimension n is called the dimension of the manifold.

Definition 12 (Atlas)**.** An **atlas** for a manifold $\mathcal{M}$ is a collection of charts $\{(U_i, \phi_i)\}$ that together cover the entire manifold. The manifold $\mathcal{M}$ is said to be **smooth** if the transition maps $\phi_i \circ \phi_j^{-1}$ are smooth for every pair i, j where $U_i \cap U_j \neq \emptyset$.

Definition 13 (Transition Functions)**.** The **transition functions** between two overlapping charts (U, ϕ) and (V, ψ) are the maps $\phi \circ \psi^{-1} : \psi(U \cap V) \to \phi(U \cap V)$ and $\psi \circ \phi^{-1} : \phi(U \cap V) \to \psi(U \cap V)$. These functions must be smooth for $\mathcal{M}$ to be a smooth manifold.

Definition 14 (Tangent Space)**.** The **tangent space** at a point p on a manifold $\mathcal{M}$, denoted as $T_p\mathcal{M}$, is a vector space that consists of the tangent vectors at p. These vectors represent the directions in which one can "move" through p, and they are formally the derivatives of curves passing through p.

Examples of Manifolds and Their Tangent Spaces:

- **Circle (S^1)**: The tangent space at any point on the circle S^1 can be visualized as the line tangent to the circle at that point. It is isomorphic to $\mathbb{R}$.
- **Sphere (S^2)**: The tangent space at any point on the sphere S^2 is the plane that touches the sphere at that point and is perpendicular to the radius at that point. It is isomorphic to $\mathbb{R}^2$.

- **Torus** ($S^1 \times S^1$): The torus, typically represented as a product of two circles, $S^1 \times S^1$, is a manifold of dimension 2. The tangent space at a point on the torus is the direct sum of the tangent spaces of the two circles that form the torus, each isomorphic to $\mathbb{R}$. Thus, it is isomorphic to $\mathbb{R}^2$.

Manifolds are fundamental in many areas of science and engineering, such as in the theory of relativity, where the universe is modeled as a four-dimensional manifold, and in robotics and computer graphics, where configurations and movements are often modeled using manifold theory.

Hyperbolic Space and Its Mathematical Structure

Hyperbolic space, denoted $\mathbb{H}^n$, is a model of n-dimensional non-Euclidean geometry characterized by a constant negative curvature. Unlike Euclidean spaces, hyperbolic spaces expand rapidly away from any given point, and the geometry of lines and triangles significantly differs.

Definition 15 (Hyperbolic Space)**.** Hyperbolic space $\mathbb{H}^n$ can be expressed as the upper half-space model in $\mathbb{R}^n$:

$$\mathbb{H}^n = \{(x_1, \ldots, x_n) \in \mathbb{R}^n : x_n > 0\} \tag{3.14}$$

equipped with the hyperbolic metric:

$$ds^2 = \frac{\sum_{i=1}^{n} dx_i^2}{x_n^2}. \tag{3.15}$$

This metric defines the infinitesimal distance between nearby points in hyperbolic space, emphasizing how distances increase as one moves towards the boundary of the half-space.

Tangent Space in Hyperbolic Space: The tangent space $T_p\mathbb{H}^n$ at any point p in hyperbolic space is crucial for defining vectors and differential operations in hyperbolic space. Given the hyperbolic metric, the inner product in the tangent space at p is given by:

$$\langle u, v \rangle_p = \frac{1}{x_n^2} \sum_{i=1}^{n} u_i v_i, \tag{3.16}$$

where $u = (u_1, \ldots, u_n)$ and $v = (v_1, \ldots, v_n)$ are vectors in the tangent space $T_p\mathbb{H}^n$.

Exponential and Logarithmic Maps in Hyperbolic Space: These mappings are crucial for modeling transitions between hyperbolic and Euclidean spaces, facilitating data representation and manipulation in various applications.

Definition 16 (Exponential Map in Hyperbolic Space)**.** The exponential map at a point p in hyperbolic space, denoted $exp_p : T_p\mathbb{H}^n \to \mathbb{H}^n$, maps a vector v in the tangent space at p to a point on the manifold. In the upper half-space model, if $p = (x_1, \ldots, x_n)$ and $v = (v_1, \ldots, v_n)$, the exponential map can be computed by:

$$\exp_p(v) = p \cdot \cosh(\|v\|) + \left(\frac{v}{\|v\|} \right) \cdot \sinh(\|v\|), \tag{3.17}$$

where $\|v\|$ is the norm of vector v calculated using the hyperbolic metric.

Definition 17 (Logarithmic Map in Hyperbolic Space)**.** Conversely, the logarithmic map at a point p in hyperbolic space, denoted $log_p : \mathbb{H}^n \to T_p\mathbb{H}^n$, maps a point q on the manifold back to a vector in the tangent space at p. For points $p = (x_1, \ldots, x_n)$ and $q = (y_1, \ldots, y_n)$, the logarithmic map is given by:

$$\log_p(q) = \frac{d(p, q)}{\tanh(d(p, q))} \cdot (q - p), \tag{3.18}$$

where $d(p, q)$ is the hyperbolic distance between p and q, calculated as:

$$d(p, q) = \operatorname{arcosh}\left(1 + \frac{2\|p - q\|^2}{x_n y_n}\right).$$

These mappings enable significant functionalities in geometric deep learning and other applications where hyperbolic geometry plays a crucial role.

Augmented Neural ODEs (ANODEs)

Neural Ordinary Differential Equations (NODEs) have shown great promise in modeling continuous-time data, attributed to their efficient memory use and elegant formulation. Despite these advantages, NODEs face a significant limitation: they can only model homeomorphic transformations, which are both continuous and invertible. This topological constraint means that NODEs preserve the topology of the input space and therefore cannot model any function that would require the "tearing apart" of the input space.

To address this limitation, Augmented Neural ODEs (ANODEs) [63] were introduced. By augmenting the system's dimensionality, ANODEs enable the representation of more complex dynamics without increasing the complexity of the ODE solver. The augmented system can be expressed mathematically as:

$$\frac{d}{dt}\begin{pmatrix}\mathbf{h}(t)\\ \mathbf{a}(t)\end{pmatrix} = \mathbf{f}\left(\begin{pmatrix}\mathbf{h}(t)\\ \mathbf{a}(t)\end{pmatrix}, t; \theta\right), \quad \begin{pmatrix}\mathbf{h}(0)\\ \mathbf{a}(0)\end{pmatrix} = \begin{pmatrix}\mathbf{x}\\ 0\end{pmatrix}, \tag{3.19}$$

where $\mathbf{h}(t) \in \mathbb{R}^d$ represents the original state, $\mathbf{a}(t) \in \mathbb{R}^p$ denotes the augmented dimensions, $\mathbf{f}$ is the neural network parameterized by θ, and $\mathbf{x}$ is the initial condition of the state.

The effectiveness of ANODEs is theoretically supported by the Whitney Embedding

Theorem [99]:

Theorem 6 (Whitney Embedding Theorem)**.** Every smooth n-dimensional manifold M can be smoothly embedded in $\mathbb{R}^{2n+1}$.

This theorem elucidates that through the utilization of extra dimensions, ANODEs are capable of learning expressive feature mappings beyond the confines of homeomorphisms, unlike traditional NODEs. Consequently, ANODEs can represent a more extensive class of functions, enhancing their performance, generalization capabilities, and computational efficiency relative to conventional NODE models.

Neural Manifold ODEs

Neural Manifold ODEs [65] adapt Neural ODEs to handle data with intrinsic geometric properties by modeling the evolution of hidden states $z(t)$ on a manifold $\mathcal{M}$ rather than in Euclidean space. This approach is particularly effective for complex data structures that are naturally represented in non-Euclidean geometries.

Neural Manifold ODEs can be expressed by the differential equation:

$$\frac{dz}{dt} = f_{\mathcal{M}}(z(t), t; \theta) \in T_{z(t)}\mathcal{M}, \tag{3.20}$$

where $f_{\mathcal{M}}$ denotes the manifold-adapted dynamics ensuring that the evolution of $z(t)$ is confined to the tangent space $T_{z(t)}\mathcal{M}$ of the manifold at $z(t)$. Here, θ represents the parameters of the neural network modeling the dynamics.

Dynamic Chart Methodology: The dynamic chart approach facilitates the integration of Neural Manifold ODEs by employing a sequence of local charts, each mapping a part of the manifold to a Euclidean space. This approach is articulated mathematically

as:

$$\text{MODE} = \varphi_k \circ \text{ODE}_k \circ (\varphi_k^{-1} \circ \varphi_{k-1}) \circ \cdots \circ (\varphi_2^{-1} \circ \varphi_1) \circ \text{ODE}_1 \circ \varphi_1^{-1}, \tag{3.21}$$

where each ODE_i represents an ODE solved in a local Euclidean space provided by the chart φ_i. The φ_i^{-1} mappings ensure correct transitions between local solutions, maintaining a coherent global solution on the manifold.

Advantages of Dynamic Charts:

- **Local Solvability:** Each segment of the ODE is solved within a local Euclidean context, which increases numerical stability and accuracy.
- **Flexibility in Transitions:** Smooth transitions between charts are managed to ensure the global coherence of the solution across the manifold.
- **Effective Utilization of Euclidean Solvers:** By transforming the manifold-bound problem into a series of local Euclidean problems, the method leverages the robustness of conventional ODE solvers adapted for Euclidean spaces.

This innovative integration technique, supported by rigorous mathematical validation, enhances the ability of Neural Manifold ODEs to model dynamics on complex geometrical structures effectively.

The exploration of Augmented Neural ODEs and Neural Manifold ODEs marks significant strides in adapting Neural ODE technology to more complex data structures. By integrating advanced mathematical concepts from manifold theory, these models not only enhance the handling of non-Euclidean geometries but also pave the way for more precise and efficient data analysis techniques. These developments highlight the ongoing shift towards incorporating geometric and topological awareness in machine learning models, promising new avenues for research and application.

3.1.4 Neural Controlled Differential Equations (NCDEs)

Neural Controlled Differential Equations (NCDEs) [72] represent another significant extension of Neural Ordinary Differential Equations (NODEs) by integrating control mechanisms that directly interact with the input data. Unlike NODEs, which generate hidden states based solely on initial conditions, NCDEs dynamically adjust the hidden state trajectory in response to irregularly sampled, continuous-time input data. This capability makes NCDEs particularly suited for modeling complex time-series data that exhibit non-uniform sampling rates.

Mathematical Background

NCDEs are formulated to handle time-series data where the input may not be regularly spaced or may come in continuous streams. The model is defined by the following differential equation:

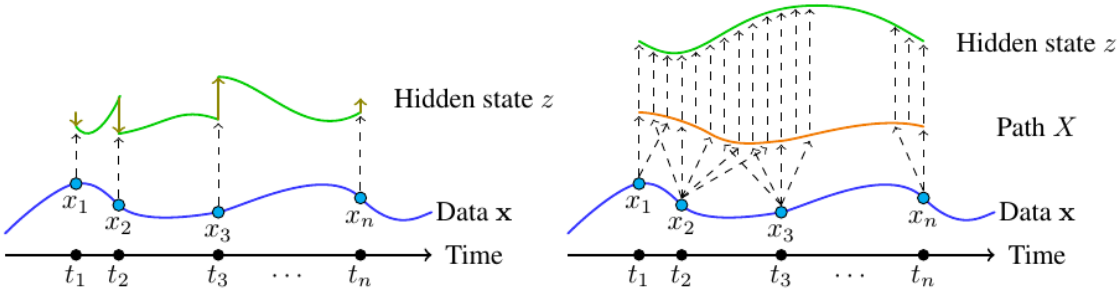
$$z(t_0) = \zeta_{\theta_2}(x_0), \quad z(t) = z(t_0) + \int_{t_0}^{t} f_{\theta_1}(z(s))dX(s), \tag{3.22}$$

where:

- $z(t)$ represents the hidden state at time t.
- $X(s)$ is a controlled path that encodes the input data up to time s.
- f_{θ_1} is a neural network that models the dynamics of the system, parameterized by θ_1.
- ζ_{θ_2} maps the initial input x_0 to the initial state $z(t_0)$, parameterized by θ_2.

This setup allows NCDEs to process input data effectively, providing a powerful framework for applications that require handling of streaming or irregularly sampled

data. The following figure [72] illustrates the process:



(a) RNNs: Interrupt the hidden state at data.

(b) Neural CDE: Continuous dependence on data.

Figure 3.2: Illustration of the Neural Controlled Differential Equation model, where the trajectory of the hidden states is influenced by the controlled input path $X(s)$.

The integration over the controlled path $X(s)$, combined with the differential component $f_{\theta_1}(z(s))$, allows the model to adjust its state dynamically in response to changes in input data, thus capturing complex dependencies and temporal characteristics of the input series.

Extension for Online Prediction

Neural Controlled Differential Equations (NCDEs) have demonstrated excellent performance in modeling functions of irregular time series in offline prediction tasks. However, existing implementations of NCDEs rely on non-causal interpolations of the data, making them unsuitable for use in online prediction tasks, where predictions need to be made in real-time.

In online prediction, the model needs to make predictions based only on the information available up to the current time point, without access to future data. This is in contrast to offline prediction, where the entire time series is observed in advance. Online prediction models are crucial for a wide range of real-world applications, such as continuous monitoring in intensive care units (ICUs), where timely decisions and interventions

are essential.

Online vs. Offline Prediction:

- Offline Prediction models are trained and make predictions based on complete datasets. They are not suitable for scenarios where data is received sequentially and predictions must be updated continually.
- Online Prediction involves updating predictions as new data arrives, without the need to retrain the model from scratch. This approach is essential for time-sensitive applications where the latency between data reception and decision-making must be minimized.

To address the limitations of existing NCDE implementations and adapt NCDEs for online prediction, [73] introduce two new control signal schemes and compare them to the previously considered options:

1. **Natural Cubic Splines**: The original control signal used in [72], which requires the full time series to be available prior to construction. This makes it unsuitable for online prediction, as the solution $z(t)$ depends on all datapoints.
2. **Linear Control**: A simple interpolation scheme that is discretely online for fully observed data, but cannot be used even discretely online in the presence of missing data.
3. **Cubic Hermite Splines with Backward Differences**: This scheme smooths the discontinuities in the linear control, while retaining the same online properties. It is discretely online and faster to integrate numerically than linear controls.
4. **Rectilinear Control**: This control signal updates the time and feature channels separately in a lead-lag fashion, resulting in a continuously online scheme. However, the parameterization is longer, leading to slower evaluation and training times.

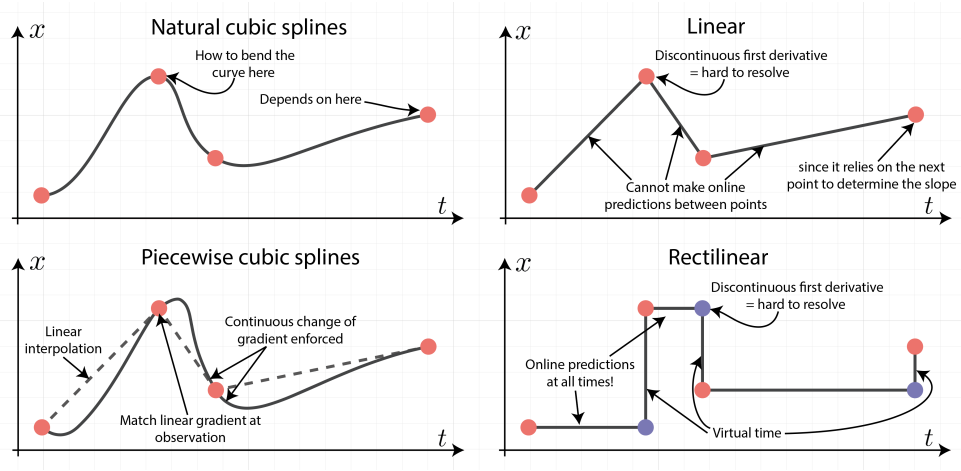


Figure 3.3: Illustration of the four control methods used in NCDEs for online prediction.

Figure 3.3 [73] shows that each method has its advantages and limitations, making them suitable for different types of applications depending on the need for real-time responsiveness and data availability .

Neural Rough Differential Equations (NRDEs)

Neural Rough Differential Equations (NRDEs) [74] generalize the concept of Neural Controlled Differential Equations (NCDEs) by leveraging the theory of signatures of rough paths. This extension allows for modeling systems driven by signals that exhibit non-smooth behavior, typical in real-world time series data. At the core of NRDEs is the approximation of the system's evolution through a Taylor expansion, using the signature of the control path to capture the intricate details within the time series.

Given an NCDE driven by a vector field f, the state update equation from $t = 0$ to some $t = s$ is expressed as:

$$Z_t = Z_0 + \int_0^t f(Z_s)\frac{dX}{dt}(s)ds \tag{3.23}$$

In NRDEs, the Taylor expansion is employed to approximate the state update equation by considering the effects of the entire subpath on the change in the system state. This is represented as follows:

$$Z_t \approx Z_0 + \int_0^t f(Z_a)\frac{dX}{dt}(s)ds + Df(Z_a)(Z_s - Z_a)\frac{dX}{dt}(s)ds \tag{3.24}$$

$$\approx Z_0 + f(Z_a)\int_0^t \frac{dX}{dt}(s)ds + Df(Z_a)\int_0^t \int_0^s f(Z_u)\frac{dX}{dt}(u)du\frac{dX}{dt}(s)ds \tag{3.25}$$

The double integral term captures the interactions between different segments of the path, which is essential for NRDEs. This term is efficiently computed using the signature of the subpath, which succinctly encapsulates the path's behavior up to a certain degree of interaction as defined in Equation 2.2.1.

Finally, the NRDE is expressed in terms of the signature as:

$$Z_t \approx Z_0 + f(Z_a)\{S(X)_{0,t}^{(i)}\}_{i=1}^d + Df(Z_a)f(Z_a)\{S(X)_{0,t}^{(i,j)}\}_{i,j=1}^d \tag{3.26}$$

Here, the signature terms $S(X)_{0,t}^{(i)}$ and $S(X)_{0,t}^{(i,j)}$ stand in for the integrals over the path increments, with the subscript $(0,t)$ indicating the integral range, while the superscript indices correspond to the dimensions of the original time series data involved in the integration. The signature-based representation elegantly handles the roughness inherent in the driving signal by considering the combined effect of the signal increments over intervals, thus generalizing the traditional NCDE model.

By utilizing rough path theory and signatures, NRDEs offer a potent framework for modeling complex dynamical systems driven by irregular data streams.

3.2 Neural Stochastic Differential Equations (NSDEs)

Stochastic differential equations (SDEs) provide a general framework for modeling dynamic systems that evolve over time and are subject to random fluctuations or noise. A typical SDE takes the form:

$$dX_t = f(X_t, t)dt + g(X_t, t)dB_t + h(X_t, t)dN_t, \tag{3.27}$$

where $f(X_t, t)$ is the drift term representing the deterministic component of the dynamics, $g(X_t, t)dB_t$ is the diffusion term modeling the continuous-time random fluctuations driven by a Brownian motion B_t, and $h(X_t, t)dN_t$ is the jump term capturing the impact of sudden events modeled by a jump process N_t.

This section explores how to extend the deterministic NODE algorithm by introducing diffusion and jump terms, leading to Neural Stochastic Differential Equations (NSDEs), to improve the NODEs model's performance and robustness and adjust it in different situations.

3.2.1 Stabilizing NODEs with Stochastic Noise

A well-known drawback of Neural Ordinary Differential Equations (ODEs) is that the system dynamics are deterministic and completely determined by the initial conditions. Small perturbations in the initial state can potentially lead to significant deviations in the final output, making the model ill-conditioned and susceptible to adversarial attacks or overfitting. To address this issue and stabilize the model's performance, [100] proposed the Neural Stochastic Differential Equation (Neural SDE) framework, which introduces stochastic noise into the continuous-time formulation.

The Neural SDE algorithm models the evolution of the hidden state $\mathbf{h}_t$ as a stochastic differential equation:

$$d\mathbf{h}_t = f(\mathbf{h}_t, t; \mathbf{w})dt + \mathbf{G}(\mathbf{h}_t, t; \mathbf{v})d\mathbf{B}_t, \tag{3.28}$$

where $\mathbf{B}_t$ is a Brownian motion, $f(\cdot)$ is a neural network parameterized by $\mathbf{w}$ (the drift term), and $\mathbf{G}(\cdot)$ is a diffusion matrix parameterized by $\mathbf{v}$ that introduces stochastic noise.

The diffusion matrix $\mathbf{G}(\mathbf{h}_t, t; \mathbf{v})$ can be designed to model various types of noise injection commonly used for regularization in discrete networks, such as additive noise, multiplicative noise, and dropout. By incorporating these stochastic regularization mechanisms into the continuous-time formulation, Neural SDEs aim to improve the generalization and robustness of Neural ODEs.

Training Neural SDEs requires developing a specialized backpropagation approach based on stochastic control theory, as outlined in [100]. The key idea is to calculate the expected loss conditioning on the initial state $\mathbf{h}_0$ and then obtain an unbiased gradient estimator using the path-wise derivative method.

The authors provide a theoretical analysis of the stability conditions of Neural SDEs, showing that the introduced stochastic noise can stabilize the dynamical system and improve robustness against input perturbations and adversarial attacks. Let $\varepsilon_t = \mathbf{h}_t^e - \mathbf{h}_t$ be the perturbation of the hidden state with $\|\varepsilon_0\| \leq \delta$, where $\mathbf{h}_t^e$ and $\mathbf{h}_t$ are the solutions of the Neural SDE with slightly different initial conditions. Furthermore, let

$$\mathbf{f}_\Delta(\varepsilon_t, t; \omega) = \mathbf{f}(\mathbf{h}_t^e, t; \mathbf{w}) - \mathbf{f}(\mathbf{h}_t, t; \mathbf{w})$$
$$\mathbf{G}_\Delta(\varepsilon_t, t; \omega) = \mathbf{G}(\mathbf{h}_t^e, t; \mathbf{w}) - G(\mathbf{h}_t, t; \mathbf{w}).$$

Then, the perturbation ε_t follows the SDE:

$$d\varepsilon_t = \mathbf{f}_\Delta(\varepsilon_t, t)dt + \mathbf{G}_\Delta(\varepsilon_t, t)d\mathbf{B}_t. \tag{3.29}$$

Definition 18 (Lyapunov stability of SDE)**.** The solution $\varepsilon_t = 0$ of 3.29:

1. is stochastically stable if for any $\alpha \in (0,1)$ and $r > 0$, there exists a $\delta = \delta(\alpha, r) > 0$ such that $\Pr\{\|\varepsilon_t\| < r \text{ for all } t \geq 0\} \geq 1 - \alpha$ whenever $\|\varepsilon_0\| \leq \delta$. Moreover, if for any $\alpha \in (0,1)$, there exists a $\delta = \delta(\alpha) > 0$ such that $\Pr\{\lim_{t\to\infty} \|\varepsilon_t\| = 0\} \geq 1 - \alpha$ whenever $\|\varepsilon_0\| \leq \delta$, it is said to be stochastically asymptotically stable;

2. is almost surely exponentially stable if $\limsup_{t\to\infty} \frac{1}{t} \log \|\varepsilon_t\| < 0$ a.s. for all $\varepsilon_0 \in \mathbb{R}^n$.

Then, the main theoretical result is given by the following theorem:

Theorem 7 (Theorem 3.2 in [100])**.** If there exists a non-negative real valued function $V(\varepsilon, t)$ defined on $\mathbb{R}^n \times \mathbb{R}_+$ that has continuous partial derivatives

$$V_1(\varepsilon, t) := \frac{\partial V(\varepsilon, t)}{\partial \varepsilon}, V_2(\varepsilon, t) := \frac{\partial V(\varepsilon, t)}{\partial t}, V_{1,1}(\varepsilon, t) := \frac{\partial^2 V(\varepsilon, t)}{\partial \varepsilon \partial \varepsilon^\top} \tag{3.30}$$

and constants $p > 0, c_1 > 0, c_2 \in \mathbb{R}, c_3 \geq 0$ such that the following inequalities hold:

- $c_1\|\varepsilon\|^p \leq V(\varepsilon, t)$
- $\mathcal{L}V(\varepsilon, t) = V_2(\varepsilon, t) + V_1(\varepsilon, t)\boldsymbol{f}_\Delta(\varepsilon, t) + \frac{1}{2}\operatorname{Tr}\left[\boldsymbol{G}_\Delta^\top(\varepsilon, t)V_{1,1}(\varepsilon, t)\boldsymbol{G}_\Delta(\varepsilon, t)\right] \leq c_2 V(\varepsilon, t)$
- $\|V_1(\varepsilon, t)\boldsymbol{G}_\Delta(\varepsilon, t)\|^2 \geq c_3 V^2(\varepsilon, t)$

for all $\varepsilon \neq \mathbf{0}$ and $t > 0$. Then for all $\varepsilon_0 \in \mathbb{R}^n$,

$$\limsup_{t\to\infty} \frac{1}{t} \log \|\varepsilon_t\| \leq -\frac{c_3 - 2c_2}{2p} \quad \text{a.s.} \tag{3.31}$$

In particular, if $c_3 \geq 2c_2$, the solution $\varepsilon_t \equiv \mathbf{0}$ is almost surely exponentially stable.

This theorem provides a theoretical justification for the improved robustness of Neural SDEs against input perturbations and adversarial attacks, as the stochastic noise can stabilize the dynamical system and prevent the amplification of small perturbations.

3.2.2 Neural Jump SDEs

While Neural SDEs can model continuous dynamics driven by Brownian motion, many real-world systems also exhibit discontinuous jumps due to discrete events. The Neural Ordinary Differential Equations (NODE) framework cannot handle such abrupt changes in the state trajectory. To address this limitation, [71] introduced Neural Jump Stochastic Differential Equations (NJSDEs) that combine the continuous dynamics of NODEs with a stochastic jump process.

In the NJSDE formulation, the state vector $\mathbf{z}_t \in \mathbb{R}^n$ evolves according to the following SDE:

$$d\mathbf{z}_t = f(\mathbf{z}_t, t; \boldsymbol{\theta})dt + w(\mathbf{z}_t, \mathbf{k}_t, t; \boldsymbol{\theta})dN_t \tag{3.32}$$

Here, $f(\mathbf{z}_t, t; \boldsymbol{\theta})$ governs the continuous dynamics as in standard NODEs, parameterized by $\boldsymbol{\theta}$. The new term $w(\mathbf{z}_t, \mathbf{k}_t, t; \boldsymbol{\theta})$ models the effect of discrete events on the state $\mathbf{z}_t$, where $\mathbf{k}_t$ represents the type or mark of the event at time t. N_t is a counting process that increments by 1 whenever an event occurs.

The events are assumed to arrive stochastically, with their conditional intensity $\lambda(\mathbf{z}_t)$ parameterized by a neural network that takes $\mathbf{z}_t$ as input. Similarly, the probability distribution $p(\mathbf{k}_t|\mathbf{z}_t)$ over event types is also modeled by a neural network conditioned on $\mathbf{z}_t$.

By combining continuous dynamics and discrete jumps in a single framework, NJSDEs can capture the behavior of hybrid systems that exhibit both smooth evolution and discontinuous changes triggered by random events. The authors derive an adjoint sen-

sitivity method to efficiently compute the gradients of the loss function with respect to the model parameters $\boldsymbol{\theta}$, enabling NJSDEs to be trained on datasets containing event sequences.

3.2.3 Neural SDEs for Data Generalization

Generative Adversarial Networks (GANs) [101] are a class of generative models that aim to learn the underlying probability distribution of the training data. A GAN consists of two neural networks, a generator G and a discriminator D, trained in an adversarial manner. The generator G takes random noise z as input and produces fake samples $G(z)$, while the discriminator D tries to distinguish between real samples from the training data and fake samples produced by the generator. The objective function for training GANs is given by:

$$\min_G \max_D \left\{ E_{x\sim P_r}[\log D(x)] + E_{z\sim P(z)}[\log(1 - D(G(z)))] \right\}, \tag{3.33}$$

where x represents samples from the real data distribution P_r, and z denotes input noise to G, aiming to model a distribution P_g that closely approximates P_r.

In [70], the authors show that Neural Stochastic Differential Equations (NSDEs) can be viewed as a continuous-time generalization of GANs. The generator in this framework is an NSDE of the form:

$$dX_t = \mu_\theta(t, X_t)dt + \sigma_\theta(t, X_t) \circ dW_t, \quad X_0 = \zeta_\theta(V) \tag{3.34}$$

where μ_θ, σ_θ, and ζ_θ are neural networks parameterized by θ, representing the drift, diffusion, and initial condition of the SDE, respectively. W_t is a Brownian motion, and V is the initial noise drawn from a standard multivariate normal distribution. The output

of the generator is then given by $Y_t = \alpha_\theta X_t + \beta_\theta$, which is a linear transformation of the hidden state X_t.

The discriminator is modeled as a Neural Controlled Differential Equation (Neural CDE) [72] with respect to the control path Y_t, which is the output of the generator:

$$dH_t = f_\phi(t, H_t)dt + g_\phi(t, H_t) \circ dY_t, \quad H_0 = \xi_\phi(Y_0) \tag{3.35}$$

where f_ϕ, g_ϕ, and ξ_ϕ are neural networks parameterized by ϕ, representing the drift, diffusion, and initial condition of the CDE, respectively. The discriminator score is then given by $D = m_\phi \cdot H_T$, where m_ϕ is a learnable vector and H_T is the terminal state of the CDE.

The authors show that by using the Wasserstein GAN objective, NSDEs can be trained to learn arbitrary stochastic processes in the infinite data limit, without the need to pre-specify statistics or density functions. They demonstrate the effectiveness of their approach on several datasets, including financial time series, air quality data, and the evolution of weights during stochastic gradient descent optimization.

This framework opens up new possibilities for generative modeling of continuous-time processes, with applications in diverse domains such as finance, biology, and physics, where stochastic differential equations are commonly used to model dynamical systems.

In recent years, diffusion models [102, 103] have emerged as a promising direction for generative modeling, and there are connections between diffusion models and NSDEs. Diffusion models can be viewed as a discretization of a certain type of SDE called the reverse-time SDE [104]. By parameterizing one of the terms in the reverse-time SDE with a neural network, diffusion models can be used to sample from complex high-dimensional distributions, such as images or audio. This has led to state-of-the-art results in various generative modeling tasks. The connection between diffusion models and NSDEs provides

a unifying framework for continuous-time generative modeling and opens up exciting avenues for future research.

3.3 Applications for physical simulation

3.3.1 Graph Neural Networks

Graph Neural Networks (GNNs) are a sophisticated class of deep learning models designed to process data that inherently possesses a graph structure. Common examples include physical systems, social networks, and biological networks. A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ comprises a set of nodes (or vertices) $\mathcal{V}$ and edges $\mathcal{E}$ that connect pairs of these nodes.

Define $N = |\mathcal{V}|$ as the number of nodes and $M = |\mathcal{E}|$ as the number of edges within the graph. Each node $v_i \in \mathcal{V}$ is associated with a feature vector $\mathbf{x}_i \in \mathbb{R}^F$ with $i = 1, \cdots, N$, where F is the feature dimensionality. Additionally, each edge $(v_i, v_j) \in \mathcal{E}$ may have an associated edge feature vector $\mathbf{e}_{ij} \in \mathbb{R}^{F'}$, where F' represents the edge feature dimensionality.

The central concept of GNNs involves learning a robust representation for each node through iterative updates based on the features of adjacent nodes. This mechanism, commonly referred to as message passing, employs both aggregation and transformation steps to refine each node's representation.

Mathematically, this message-passing process in GNNs can be articulated through the following equations:

$$\mathbf{m}_i^{(k+1)} = \sum_{j \in \mathcal{N}(i)} \mathrm{MSG}^{(k)}(\mathbf{h}_i^{(k)}, \mathbf{h}_j^{(k)}, \mathbf{e}_{ij}; \theta_{\mathrm{msg}}) \tag{3.36}$$

$$\mathbf{h}_i^{(k+1)} = \mathrm{UPDATE}^{(k)}(\mathbf{h}_i^{(k)}, \mathbf{m}_i^{(k+1)}; \theta_{\mathrm{upd}}) \tag{3.37}$$

Here, $\mathbf{h}_i^{(k)}$ denotes the representation of node v_i at the k-th iteration. The function $\mathcal{N}(i)$ represents the set of neighbors of node v_i. The functions $\text{MSG}^{(k)}$ and $\text{UPDATE}^{(k)}$ compute the messages and update the node representations, respectively, and are parameterized by neural networks with parameters θ_{msg} and θ_{upd}.

After K iterations of message passing, the representations $\mathbf{h}_i^{(K)}$ are leveraged for downstream tasks such as node classification, link prediction, or graph classification.

GNNs have demonstrated exceptional utility in a range of applications, particularly in simulating physical systems, where they effectively model complex dynamics encoded in graph structures. In this section, we will see the integration of GNNs with Neural Differential Equations (NDEs) for advanced physical simulations, showcasing their combined strength in capturing both spatial relationships and continuous-time dynamics.

3.3.2 HOPE: High-Order Graph ODE for Modeling Interacting Dynamics

Motivation

While dynamic interacting systems are prevalent and influential in both natural and social sciences, effectively modeling their complexities presents substantial challenges. Traditional approaches using Graph Neural Networks (GNNs) primarily handle object interactions within a static or discretely evolving graph framework, focusing on learning representations at distinct time steps. These methods, however, struggle with irregularly sampled data and require complete observations at each timestep, which is often impractical in real-world scenarios.

The integration of Neural Ordinary Differential Equations (ODEs) with GNNs has offered a pathway to model system dynamics continuously, which allows for handling missing data and capturing smoother system evolutions. Despite these advancements,

current implementations predominantly employ first-order differential equations which model the rate of change in state but overlook the acceleration, a crucial aspect in many physical and social phenomena described by second-order dynamics, such as planetary motion or oscillatory systems.

Furthermore, existing graph ODE frameworks suffer from several limitations:

- **Insufficient Capture of High-Order Correlations:** Most current models utilize spatial-based GNNs within their encoders, constructing only temporal graphs without adequately addressing the complex, non-linear interactions that span beyond immediate neighbors in the graph structure. This oversight leads to a failure in capturing high-order dependencies that are pivotal for understanding the full spectrum of interactions within dynamic systems.

- **Inefficiency and Inadequacy of First-Order Models:** The reliance on first-order derivatives not only restricts the model's capacity to express more complex natural laws but also results in inefficiencies during training and inference. These models require a high number of function evaluations (NFEs) to achieve accurate results, which significantly hampers their scalability and practicality in larger systems.

In response to these challenges, our work introduces the High-Order graPh ODE (HOPE) [105], a novel approach that leverages both advanced graph encoding techniques and second-order differential equations. HOPE is designed to initialize and evolve state representations through a twin graph encoder using dual GNN branches. This setup captures both spatial and temporal correlations effectively. Additionally, by incorporating second-order dynamics, HOPE not only aligns more closely with physical laws governing many dynamic systems but also improves the efficiency of learning by reducing the required NFEs and enhancing model convergence rates. This approach allows for a

deeper understanding and more accurate forecasting of complex dynamics in evolving graphs, pushing the boundaries of what can be achieved with neural network models in this domain.

Methodology

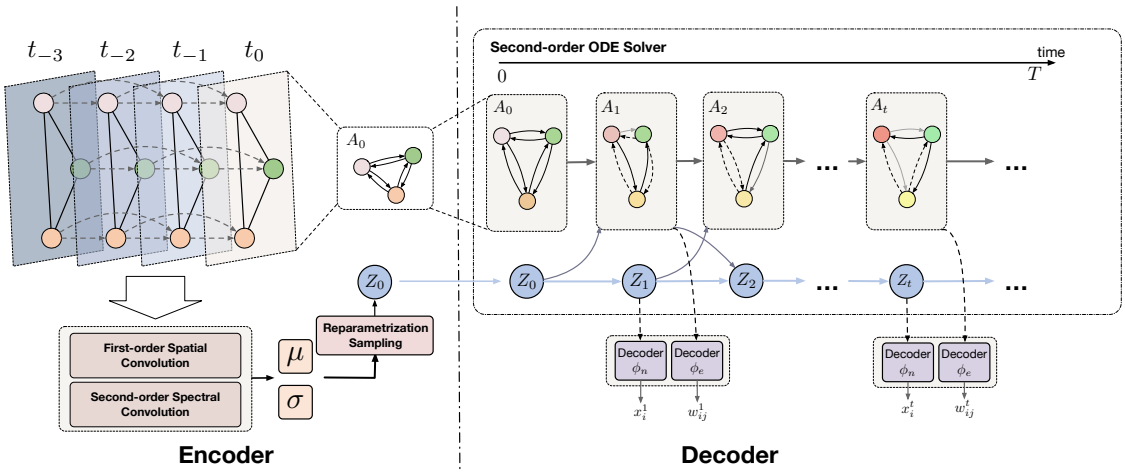


Figure 3.4: Overview of our proposed HOPE. To begin, the twin encoder utilizes two branches of graph convolution to extract spatio-temporal relationships for the initialization of latent state representations. Then, a generative model utilizes second-order ODEs to simulate the evolution of both nodes and edges. In the end, HOPE feeds state representations into the decoders to output the predicted nodes and edges. We maximize the evidence lower bound (ELBO) of the likelihood during optimization.

In the HOPE model, we propose an innovative approach to model dynamic systems by enhancing the learning of high-order spatio-temporal correlations, thereby improving both model capacity and optimization efficiency. Our model consists of three main components: (1) a twin graph encoder for initializing state representations, (2) a continuous-time generative model utilizing second-order graph ODEs, and (3) a decoder for outputting prediction values. Detailed architecture can be found in Figure 3.4.

Twin Graph Encoder The twin graph encoder in HOPE is designed to capture complex spatio-temporal relationships, initializing latent states for objects and edges through

two specialized GNN branches that process different aspects of graph data:

- **Temporal Graph Construction:** Constructs a temporal graph $\mathcal{G}^H$ that accounts for both spatial and temporal correlations. Nodes represent object observations over time, connected by spatial and temporal edges formulated as:

$$\mathbf{A}(i_t, j_{t'}) = \begin{cases} w_{ij}^t & t = t' \\ 1 & i = j, t' = t + 1 \\ 0 & \text{otherwise} \end{cases} . \tag{3.38}$$

- **Graph Convolution Operations:** Utilizes first-order spatial convolution integrated with a self-attention mechanism to adaptively learn from neighborhood data. Meanwhile, second-order spectral graph convolution is employed to handle non-neighborhood interactions, enhancing the encoder's ability to capture deeper semantic relationships buried within the graph spectrum.

Node Representation Learning: Node features are processed through layers of graph convolutions, refining the embeddings based on both immediate and extended spatial relations, ensuring a comprehensive initialization of object and edge states which are critical for the subsequent generative modeling:

$$\mathbf{h}_i^{t,(k+1)} = \mathbf{h}_i^{t,(k)} + \sigma\left(\sum_{j \in \mathcal{N}(i)} s^{(k)}(v_i^t, v_j^{t'}) \mathbf{W}_{value} \mathbf{h}_j^{t',(k)}\right), \tag{3.39}$$

where $s^{(k)}(v_i^t, v_j^{t'})$ represents interaction scores between each central node and its neighbors which is calculated by

$$s^{(k)}(v_i^t, v_j^{t'}) = \mathbf{A}(i_t, j_t) cos(\mathbf{W}_{query} \mathbf{h}_i^{t,(k)}, \mathbf{W}_{key} \mathbf{h}_j^{t',(k)}).$$

$\mathcal{N}(i)$ denotes the neighbors of node i, $\mathbf{W}$ are the weight matrices, k is the number of iteration and σ is a non-linear activation function.

Sequence Representation Learning: Combines spatial and spectral representations, employing an attention mechanism to aggregate these into a coherent sequence representation for each object. This representation is used to initialize the latent states in the generative model:

$$\mathbf{u}_i = \frac{1}{T}\sum_{t=1}^{T}\sigma(\alpha_i^t \mathbf{q}_i^t), \tag{3.40}$$

where α_i^t are attention weights derived from the sequence of node representations, providing a dynamic summarization of the temporal interactions. And $\mathbf{q}_i^t$ are the combined representations calculated by:

$$\mathbf{q}_i^t = \delta([\mathbf{e}_i^t, \mathbf{h}_i^t]) + TE(t) \tag{3.41}$$

This encoder structure not only prepares detailed state initializations for the model's dynamical simulations but also directly addresses the challenges in capturing high-order correlations within dynamic systems.

Neural Coupled Graph ODE Model With the initial latent state representations for nodes and edges established, we introduce a neural graph ODE model to simulate the dynamics of interacting systems in a generative manner. Traditional models typically utilize first-order ODEs, which are not efficient for capturing long-term dependencies and require a large number of function evaluations (NFEs) for both optimization and inference, leading to inefficiencies. To overcome these limitations, we employ second-order neural ODEs, which align more closely with physical laws in complex systems and have been shown to achieve faster convergence in numerical optimization.

Formulation of the Second-Order Graph ODE We define the dynamics of the node states with a second-order graph ODE function as follows:

$$\frac{d^2\mathbf{Z}^t}{dt^2} + \gamma\frac{d\mathbf{Z}^t}{dt} = \sigma((\hat{\mathbf{D}}^t)^{-1}\hat{\mathbf{A}}^t\mathbf{Z}^t\mathbf{W}_p) - \mathbf{Z}^t, \tag{3.42}$$

where $\mathbf{Z}^t$ represents the latent state matrix at time t, and γ is a damping coefficient that balances the influence of first-order and second-order derivatives, facilitating a more robust simulation of dynamic processes.

Edge Dynamics To update the adjacency matrix $\hat{\mathbf{A}}^t$ with dynamic edge information, we incorporate a similar second-order differential equation:

$$\frac{d^2\mathbf{\Pi}_{ij}^t}{d^2t} + \gamma\frac{d\mathbf{\Pi}_{ij}^t}{dt} = \rho_n\left(\left[\mathbf{z}_i^t\|\mathbf{z}_j^t\|\mathbf{z}_i^t\odot\mathbf{z}_j^t\right]\right) + \rho_e\left(\mathbf{\Pi}_{ij}^t\right), \tag{3.43}$$

where ρ_n and ρ_e are MLPs that process node and edge information, respectively.

Optimization with Momentum The update rule for the ODEs utilizes a momentum-based optimization approach, extending classical momentum techniques to the graph domain:

$$\mathbf{Z}^{t+1} = (1-\lambda)\mathbf{Z}^t + \lambda\sigma((\hat{\mathbf{D}}^t)^{-1}\hat{\mathbf{A}}^t\mathbf{Z}^t\mathbf{W}_p) + \beta\left(\mathbf{Z}^t - \mathbf{Z}^{t-1}\right), \tag{3.44}$$

where λ and β control the learning rate and momentum terms, respectively. This method significantly enhances the convergence rate by incorporating information about previous updates.

Lemma 1. Given the momentum updating algorithm for GNNs, we have Equation 3.42 when $\lambda \to 0$.

Lemma 2. Our ODE formalization can be transformed into an augmented first-order ODE.

Lemma 3. Given the initial state $(t_0, \mathbf{Z}^{t_0})$, we claim that there exists $\varepsilon > 0$, s.t. Equation 3.42 has a unique solution in the interval $[t_0 - \varepsilon, t_0 + \varepsilon]$ when the activation function $\sigma(\cdot)$ is ReLU.

The proofs of these lemmas can be found in Appendix A (See proof A.1.1, A.1.2, A.1.3). This methodology ensures not only theoretical robustness but also practical effectiveness, as demonstrated in our experiments.

Decoder and Optimization The decoding and optimization stages are crucial for reconstructing the input data from the learned latent states and refining the overall predictive performance of the model. We utilize two decoders in our framework, each specialized for a different type of output based on the latent states:

$$\mu_i^t = \phi_n(\mathbf{z}_i^t); \quad \mu_{ij}^t = \phi_e(\mathbf{\Pi}_{ij}^t), \tag{3.45}$$

where ϕ_n and ϕ_e are distinct MLPs that respectively decode the node and edge information from their corresponding latent representations.

In training, our model employs the variational inference technique to optimize the evidence lower bound (ELBO), enhancing the likelihood of observed data while ensuring a minimal divergence between the prior and the inferred posterior distributions:

$$\ell_{ELBO} = \mathbb{E}_{Z^0 \sim \prod_{i=1}^N q\left(z_i^0 | \mathcal{X}, \mathcal{A}\right)} \left[\log p(\mathcal{X}, \mathcal{A})\right] - \mathrm{KL}\left[\prod_{i=1}^N q(\mathbf{z}_i^0 | \mathcal{X}, \mathcal{A}) \| p\left(\mathbf{Z}^0\right)\right], \tag{3.46}$$

where $KL(\cdot||\cdot)$ represents the Kullback-Leibler divergence. This optimization process is computed individually for each node and edge, allowing for a precise adjustment to the

model parameters based on the reconstruction error:

$$\ell_{ELBO} = -\sum_i \sum_t \frac{\|\mathbf{x}_i^t - \mu_i^t\|^2}{2\sigma^2} - \sum_i \sum_j \sum_t \frac{\left\|w_{ij}^t - \mu_{ij}^t\right\|^2}{2\sigma^2} - \mathrm{KL}\left[\prod_{i=1}^{N} q(\mathbf{z}_i^0|\mathcal{X}, \mathcal{A}) \| p\left(\mathbf{Z}^0\right)\right], \tag{3.47}$$

where σ^2 denotes the variance of the prior distribution, ensuring that the model not only predicts accurately but also adheres closely to the distributional assumptions encoded by the prior. The detailed algorithm and additional implementation details are provided in the Appendix.

Experiment Results

Datasets. We evaluated our model using three key datasets:

- **COVID-19**: Daily trend data sourced from the Johns Hopkins University Center for Systems Science and Engineering.
- **Social Network**: Simulates opinion dynamics within a social network.
- **Spring Oscillator**: Models physical dynamics of a system of interconnected springs and balls.

Baseline Comparison. Our approach is compared against various established methods, including LSTM, GRU, DGCRN for neural network models, and NODE, HBNODE, MPNODE, and CG-ODE for neural ODE-based models.

For more details about dataset and baseline models, please see appendix B.

Implementation Details. The model was implemented using Pytorch and optimized with Adam over 100 epochs. We used a single-layer graph convolutional network with second-order ODE integration in our model encoder.

Evaluation Metrics. Performance was assessed using MAE, RMSE, and MAPE, with MAPE excluded for the Spring Oscillator due to near-zero ground truth values.

Table 3.1: Results of compared methods on COVID-19 with the prediction length one week, two weeks and three weeks. **Bold** numbers indicate the best performance whereas underline numbers indicate the second best performance.

Methods	1-week-ahead			2-week-ahead			3-week-ahead			Average		
	MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE
LSTM	231.7	370.4	0.0816	482.3	771.8	0.1608	547.5	865.9	0.1809	420.5	669.4	0.1411
GRU	234.9	373.8	0.0786	491.8	780.3	0.1544	555.3	874.0	0.1739	427.3	676.0	0.1356
NODE	103.1	184.7	0.0349	183.7	293.4	0.0516	255.7	400.1	0.0677	180.8	292.7	0.0514
HBNODE	243.8	383.6	0.0712	220.1	380.8	0.0527	283.3	464.2	0.0767	249.1	409.5	0.0669
DGCRN	102.4	161.5	0.0279	191.7	301.0	0.0479	281.4	428.1	0.0739	191.8	296.9	0.0499
MPNODE	152.7	237.5	0.0357	272.0	549.4	0.0527	248.7	385.8	0.0696	224.5	390.9	0.0527
CG-ODE	91.59	146.9	0.0255	182.5	298.9	0.0431	251.1	385.6	0.0632	175.1	277.1	0.0439
Ours	**85.64**	**146.0**	**0.0228**	**180.9**	**275.2**	**0.0397**	**243.1**	**373.3**	**0.0612**	**169.9**	**264.8**	**0.0412**

Table 3.2: Results of compared methods on Social Network with the prediction length 10, 20 and 40.

Methods	10			20			40			Average		
	MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE
LSTM	0.0933	0.1367	0.1269	0.1880	0.2719	0.2748	0.3677	0.5268	0.6276	0.2163	0.3118	0.3431
GRU	0.0939	0.1374	0.1280	0.1844	0.2724	0.2772	0.3916	0.5478	0.6566	0.2233	0.3192	0.3539
NODE	0.1276	0.1630	0.1161	0.2430	0.3066	0.2521	0.5383	0.6742	0.6025	0.3030	0.3813	0.3236
HBNODE	0.1230	0.1596	0.1149	0.2701	0.3358	0.2635	0.4877	0.6247	0.5934	0.2936	0.3734	0.3239
DGCRN	0.0913	0.1284	0.1192	0.2337	0.3046	0.2558	0.4764	0.5723	0.5614	0.2671	0.3351	0.3121
MPNODE	0.0887	0.1198	0.1151	0.1792	0.2554	0.2549	0.3678	0.5239	0.5214	0.2119	0.2997	0.2971
CG-ODE	0.0852	0.1236	0.1205	0.2073	0.2436	0.2432	0.3199	**0.4871**	0.4909	0.2041	0.2848	0.2849
Ours	**0.0796**	**0.1167**	**0.1050**	**0.1543**	**0.2203**	**0.2174**	**0.3019**	0.4873	**0.4867**	**0.1786**	**0.2748**	**0.2697**

Table 3.3: Results of compared methods on Spring Ocsillator with the prediction length 36, 48 and 60.

Methods	36		48		60	
	MAE	RMSE	MAE	RMSE	MAE	RMSE
LSTM	0.2661	0.3401	0.4120	0.5321	0.6257	0.7775
GRU	0.3110	0.3910	0.3940	0.4997	0.6369	0.7950
NODE	0.2757	0.3491	0.4569	0.5702	0.6259	0.7770
HBNODE	0.2969	0.3792	0.4512	0.5778	0.6486	0.8086
DGCRN	0.2798	0.3623	0.4013	0.5296	0.6159	0.7683
MPNODE	0.3473	0.4340	0.4161	0.5199	0.6133	0.7648
CG-ODE	0.2723	0.3515	0.4178	0.5382	0.6180	0.7729
Ours	**0.2649**	**0.3387**	**0.3329**	**0.4351**	**0.5883**	**0.7359**

Performance Tables The comparative performance of our model against baseline methods is detailed in Tables 3.1, 3.2, and 3.3, demonstrating superior accuracy and efficiency across all datasets.

Efficiency Analysis Figure 3.5 shows the number of function evaluations (NFE) needed for our model compared to CG-ODE, indicating higher efficiency and faster convergence.

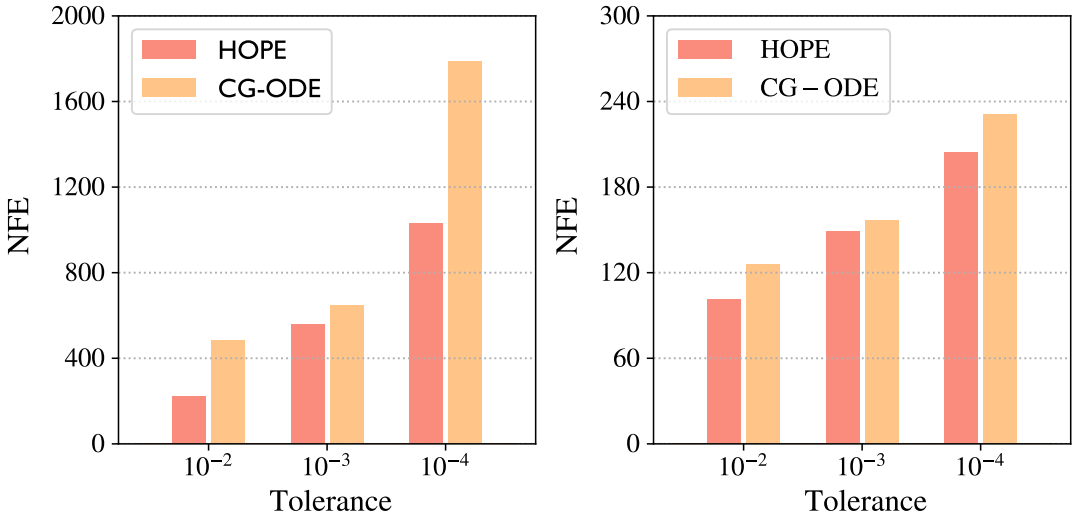


Figure 3.5: Comparison of NFEs between CG-ODE and our model on the COVID-19 and Social Network datasets.

Table 3.4: Ablation study on Social Network.

Methods	10			20			40			Avgence		
	MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE
HOPE w.o. FC	0.09910	0.1222	0.1172	0.1856	0.2669	0.2624	0.3934	0.5360	0.5421	0.2260	0.3084	0.3072
HOPE w.o. FO	0.08477	0.1334	0.1237	0.2018	0.2761	0.2835	0.3244	0.5163	0.4935	0.2037	0.3086	0.3002
HOPE w.o. SC	0.08322	0.1169	0.1122	**0.1520**	0.2384	0.2682	0.3167	**0.4700**	0.5782	0.1840	0.2751	0.3195
HOPE w.o. SO	0.08067	0.1212	0.1099	0.1793	0.2497	0.2223	0.3029	0.4847	0.6470	0.1876	0.2852	0.3264
HOPE w.o. E	0.09612	0.1358	0.1249	0.1879	0.2648	0.2616	0.3612	0.6249	0.5238	0.2151	0.3418	0.3034
Ours	**0.07958**	**0.1167**	**0.1050**	0.1543	**0.2203**	**0.2174**	**0.3019**	0.4873	**0.4867**	**0.1786**	**0.2748**	**0.2697**

Ablation Study An ablation study (Table 3.4) was conducted to verify the contribution of each model component, affirming the necessity of our model's architectural choices.

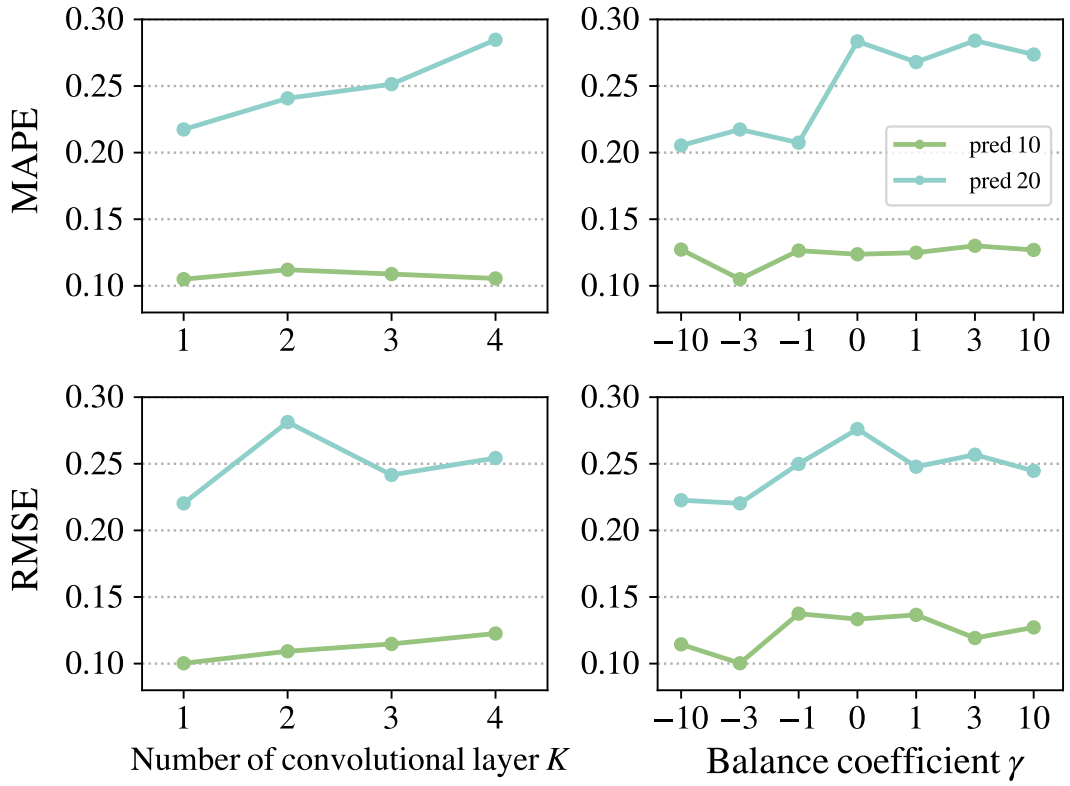


Figure 3.6: Sensitivity analysis of hyperparameters K and γ on prediction performance in the Social Network dataset.

Sensitivity Analysis Figure 3.6 explores how different settings of K and γ impact model performance, aiding in the optimal configuration of our model parameters.

3.3.3 CARE: Modeling Interacting Dynamics Under Temporal Distribution Shift

Motivation

Modeling interacting dynamical systems accurately is crucial for applications ranging from social network analysis to physical simulations. These systems are inherently complex, with interactions that can be represented using geometric graphs where nodes and edges depict objects and their interactions, respectively. Graph Neural Networks (GNNs) have shown promising results in capturing these interactions by predicting object trajectories based on the aggregated information from their neighbors.

However, a significant challenge arises when dealing with long-term dependencies and temporal distribution shifts in dynamic environments. Traditional GNN-based methods are adept at next-step predictions but struggle with error accumulation over extended periods, especially under conditions where the system's environmental or relational structures change. Such changes can be triggered by factors like varying temperatures or pressures, which in turn can cause a continuous temporal distribution shift, making the prediction task even more daunting.

Most existing methods focus on stable, in-distribution trajectories and fail to generalize well to out-of-distribution data, which is often encountered in real-world scenarios due to continuous shifts. Unlike discrete domain shifts commonly addressed in vision and text applications, interacting dynamical systems experience continuous shifts that require models to generalize across varying conditions effectively.

This motivation leads us to develop a model that not only predicts with high accuracy under known conditions but also adapts effectively to changes, capturing the continuous and often unpredictable variations in dynamical systems. Our proposed model, Context-attended Graph ODE (CARE) [106], is designed to handle these challenges by

incorporating context variables that capture the essence of temporal distribution shifts, thereby enhancing the model's ability to generalize across different dynamic states.

Methodology

In the methodology of CARE, we introduce a sophisticated approach to model interacting dynamics under temporal domain shifts through a context-attended graph ODE system. This approach systematically incorporates assumptions and lemmas crucial for understanding how temporal distribution shifts affect system behavior.

Formulating the Coupled ODE System To capture the dynamics of nodes and contexts under changing conditions, we make foundational assumptions:

Assumption 8. (Independence-I) The context variable is independent of the sequences before the last observed timestamp, i.e., $P(\mathbf{c}^t|\mathbf{c}^{t-k}, G^{0:t}) = P(\mathbf{c}^t|\mathbf{c}^{t-k}, G^{t-k:t})$, where $t-k$ is the last observed timestamp.

In the assumption, $\mathbf{c}^t$ is a context variable. For example, the context variable could indicate flow speed, density and viscosity in fluid dynamics. G^t represents the graph state at time t, and the abbreviation $G^{0:t} = \{G^0, \cdots, G^t\}$ represents the sequence of graph states for convenience.

Assumption 9. (Independence-II) Given the current states and contexts, the future trajectories are independent of the previous trajectories and contexts, i.e.,

$$P(\mathbf{Y}^{t-k:t+l}|G^{0:t-k}, \mathbf{c}^{0:t-k}) = P(\mathbf{Y}^{t-k:t-k+l}|G^{t-k}, \mathbf{c}^{t-k})$$

where l is the length of the prediction.

Then, we can have the following lemma:

Lemma 4. With Assumptions 8 and 9, we have:

$$\begin{aligned} \mathrm{P}\left(\mathbf{Y}^t \mid G^{0:t-1}\right) = \int \mathrm{P}\left(\mathbf{Y}^t \mid \mathbf{c}^{t-1}, G^{t-1}\right) \cdot \\ \mathrm{P}\left(\mathbf{c}^{t-1} \mid \mathbf{c}^{t-k}, G^{t-k:t-1}\right) \cdot \mathrm{P}\left(\mathbf{c}^{t-k} \mid G^{0:t-k}\right) d\mathbf{c}^{t-1} d\mathbf{c}^{t-k}. \end{aligned} \tag{3.48}$$

Graph ODE Modeling for Continuous Dynamics To model continuous evolution effectively, we incorporate an ODE system into CARE. The precondition for applying neural ODE models, which are ideal for dynamic systems, requires that both the context variable and node representations be continuously differentiable:

Assumption 10. We assume that both context variable $\mathbf{c}^s$ and node representations $\mathbf{v}_i^s$ are continuously differentiable with respect to s.

We utilize this continuous framework to define the dynamics of node states and the context variable, facilitating the integration of state changes over time:

$$\frac{d\mathbf{v}_i^s}{ds} = \Phi([\mathbf{v}_1^s, \cdots, \mathbf{v}_N^s, \mathbf{c}^s]) = \sigma\Big(\sum_{j \in \mathcal{N}^s(i)} \frac{\hat{\mathbf{A}}_{ij}^s}{\sqrt{\hat{D}_i^s \cdot \hat{D}_j^s}} \mathbf{v}_j^s \mathbf{W}_1 + \mathbf{c}^s \mathbf{W}_2\Big), \tag{3.49}$$

where $\hat{\mathbf{A}}^s$ denotes the adjacency matrix at timestamp s with self-loop, $\hat{D}_i^s$ represents the degree of node i, and $\mathcal{N}^s(i)$ includes the neighbors of node i at that timestamp.

$$\frac{d\mathbf{c}^s}{ds} = \Phi^c(\mathrm{AGG}(\{\mathbf{v}_i^s\}_{i \in V}), \mathrm{AGG}(\{\frac{d\mathbf{v}_i^s}{ds}\}_{i \in V}), \mathbf{c}^s), \tag{3.50}$$

where Φ^c is an MLP with the concatenated input and $\mathrm{AGG}(\cdot)$ is an operator to summarize node representations such as averaging and sum.

Assumption 11. All time-dependent coefficients in Eqn. 3.49, i.e., $\mathbf{A}_{ij}^t, \hat{D}_i^t$ are continuous with respect to t and bounded by a constant $C > 0$. All parameters in the weight matrix are also bounded by a constant $W > 0$.

To simplify the analysis, we set AGG(·) to summation and rewrite Eqn. 3.50 with learnable matrices $\mathbf{W}_3$, $\mathbf{W}_4$ and $\mathbf{W}_5$ as:

$$\frac{d\mathbf{c}^s}{ds} = \sigma\left(\sum_{i=1}^{N}(\mathbf{v}_i^s\mathbf{W}_3 + \frac{d\mathbf{v}_i^s}{ds}\mathbf{W}_4) + \mathbf{c}^s\mathbf{W}_5\right). \tag{3.51}$$

Then, with Assumption 11, we can deduce the following lemma:

Lemma 5. Given the initial state $(t_0, \mathbf{v}_1^{t_0}, \cdots, \mathbf{v}_N^{t_0}, \mathbf{c}^{t_0})$, we claim that there exists ε ¿ 0, such that the ODE system 3.49 and 3.50 has a unique solution in the interval $[t_0-\varepsilon, t_0+\varepsilon]$.

A comprehensive theoretical analysis supporting the model's predictive capabilities, based on historical data, is provided in the Appendix A. This includes the proofs for lemma 4 and 5 (See proof A.2.1, A.2.2 respectively).

Experiment Results

We conducted extensive evaluations of the CARE model on both particle-based and mesh-based physical systems to assess its efficacy under various simulation conditions, emphasizing its robustness to temporal distribution shifts.

Data Split and Training Details A rigorous data split strategy was implemented, with 80% of the data reserved for training and 10% each for validation and testing. During training, each trajectory sample was divided into a conditional part for initializing node and context representations, and a prediction part for model supervision, facilitating accurate assessment over varying prediction lengths.

Baseline Comparison CARE was compared against several state-of-the-art models such as LSTM, STGCN, GNS, MeshGraphNet, TIE, CG-ODE, and MP-NODE. This

comparison highlighted CARE's advancements in handling complex temporal interactions and distribution shifts.

Performance on Particle-based Physical Simulations CARE was tested on *Lennard-Jones Potential* and *3-body Stillinger-Weber Potential*, both sensitive to environmental conditions like temperature changes, to predict future velocities in all directions (v_x, v_y, and v_z).

Table 3.5: The RMSE ($\times 10^{-2}$) results of the compared methods with the prediction lengths 1, 5, 10 and 20. v_x, v_y and v_z represent the velocity in the direction of each coordinate axis.

Prediction Length	+1			+5			+10			+20		
Variable	v_x	v_y	v_z	v_x	v_y	v_z	v_x	v_y	v_z	v_x	v_y	v_z
Lennard-Jones Potential												
LSTM	3.95	3.92	3.68	9.12	9.21	9.15	10.84	10.87	10.76	14.82	14.94	14.67
GNS	3.28	3.75	3.39	7.97	8.05	7.68	10.09	10.15	10.13	13.65	13.62	13.59
STGCN	2.91	3.08	2.95	5.06	5.17	5.11	6.89	6.90	6.93	9.31	9.32	9.44
MeshGraphNet	2.89	3.13	2.94	5.29	5.53	5.28	7.03	7.09	7.11	9.12	9.21	9.24
CG-ODE	1.79	2.05	1.71	3.47	3.92	3.38	5.46	5.99	5.36	9.03	9.26	8.92
TIE	1.62	1.98	1.47	3.25	3.90	3.15	5.24	5.82	5.17	8.24	8.34	8.47
Ours	**0.76**	**0.89**	**1.01**	**2.94**	**3.16**	**2.85**	**5.01**	**4.69**	**4.71**	**5.75**	**5.91**	**5.82**
3-body Stillinger-Weber Potential												
LSTM	17.11	17.14	17.18	23.64	23.69	23.60	25.46	25.42	25.48	28.44	28.45	28.44
GNS	15.39	15.27	15.33	22.14	22.19	22.17	25.29	25.36	25.31	27.18	27.15	27.14
STGCN	12.33	12.31	12.35	17.94	17.96	17.91	20.08	20.14	20.13	23.49	23.51	23.52
MeshGraphNet	12.16	12.10	12.13	18.33	18.38	18.34	20.65	20.62	20.71	23.62	23.54	23.61
CG-ODE	9.78	9.74	9.75	12.11	12.05	12.14	15.55	15.58	15.50	16.17	16.24	16.22
TIE	10.18	10.26	10.19	14.75	14.70	14.73	18.42	18.45	18.41	20.92	21.04	21.36
Ours	**4.21**	**4.29**	**4.18**	**9.74**	**9.79**	**9.71**	**13.65**	**13.71**	**13.57**	**15.30**	**15.39**	**15.35**

From table 3.5, we can see CARE significantly outperformed all baselines, achieving

error reductions of 24.03% and 36.35% on the two datasets, respectively, attributable to the effective use of context variables and robust learning techniques.

Performance on Mesh-based Physical Simulations CARE was evaluated on *CylinderFlow* and *Airfoil*, involving complex fluid dynamics with cyclically varying flow conditions.

Table 3.6: The RMSE results of the compared methods over different prediction lengths 1, 10, 20 and 50. v_x, v_y and p represent the velocity in different directions and the pressure field, respectively.

Prediction Length	+1			+10			+20			+50		
Variable	v_x	v_y	p	v_x	v_y	p	v_x	v_y	p	v_x	v_y	p
CylinderFlow												
LSTM	3.35	29.4	12.5	7.06	44.8	17.8	9.47	49.5	19.9	14.3	73.6	42.3
GNS	3.12	28.8	11.9	7.18	44.3	17.3	9.01	49.6	19.2	13.5	73.2	41.6
STGCN	2.68	26.7	11.0	5.47	42.1	16.9	6.72	45.6	18.4	9.15	68.7	40.0
MeshGraphNet	1.75	22.4	10.6	4.09	39.7	15.7	5.38	44.5	17.2	7.92	64.3	37.7
CG-ODE	1.05	20.4	8.51	3.44	36.8	13.6	4.15	38.5	17.1	5.14	61.2	32.3
TIE	1.22	20.8	8.94	3.75	35.2	13.0	4.62	40.6	16.0	5.87	59.5	32.1
Ours	**0.87**	**19.1**	**7.21**	**3.02**	**32.9**	**11.8**	**3.95**	**37.8**	**13.9**	**4.97**	**55.8**	**29.4**
Airfoil												
LSTM	7.49	7.73	1.92	8.86	9.02	3.78	10.8	11.0	4.71	14.9	15.7	4.96
GNS	6.95	7.14	1.69	8.20	8.34	3.34	10.2	10.5	3.98	14.2	14.1	4.11
STGCN	6.24	5.35	1.07	6.57	6.51	2.33	7.88	8.01	3.16	11.6	11.8	3.17
MeshGraphNet	4.72	4.68	0.50	5.89	5.74	1.23	6.32	6.48	1.85	9.03	9.12	2.08
CG-ODE	4.26	4.32	0.35	4.78	4.70	0.46	5.81	5.66	1.04	7.39	7.85	1.69
TIE	4.17	4.39	0.33	4.99	4.86	0.51	5.75	5.62	0.95	7.25	7.63	1.44
Ours	**3.51**	**4.11**	**0.19**	**3.86**	**3.75**	**0.34**	**4.16**	**4.12**	**0.45**	**6.74**	**6.82**	**0.81**

In table 3.6, CARE showed superior performance, exceeding the best baseline by

12.99% and 22.78% on the respective datasets. The model's effectiveness in simulating unsteady flow dynamics was particularly notable.

Further Analysis Ablation Study. To further validate the effectiveness of different components within CARE, we conducted an ablation study, removing key features such as the context variable and robust learning term to assess their individual impacts on model performance. The results of this study are detailed in the following table:

Table 3.7: Ablation study on four datasets.

Datasets	Lennard-Jones			3-body Stillinger-Weber			CylinderFlow			Airfoil		
Variable	v_x	v_y	v_z	v_x	v_y	v_z	v_x	v_y	p	v_x	v_y	p
CARE V1	6.98	7.12	7.06	18.2	18.3	18.3	6.13	60.4	32.2	7.13	7.21	1.43
CARE V2	6.03	6.35	6.30	16.8	16.5	16.6	5.21	57.2	29.8	6.94	6.99	1.15
Ours	**5.75**	**5.91**	**5.82**	**15.3**	**15.4**	**15.4**	**4.97**	**55.8**	**29.4**	**6.74**	**6.82**	**0.81**

Table 3.7 demonstrate that removing the context variable (CARE V1) or the robust learning term (CARE V2) leads to a noticeable degradation in performance, underscoring their significance in enhancing the model's ability to adapt to environmental changes and distribution shifts.

Parameter Sensitivity. We also explored how different settings for condition lengths and prediction lengths affect model performance. This sensitivity analysis helps in understanding the optimal configuration for both short-term and long-term forecasting accuracy. The results, as illustrated in the following figure, show that longer condition lengths generally improve model performance, likely due to the increased amount of information available for making predictions:

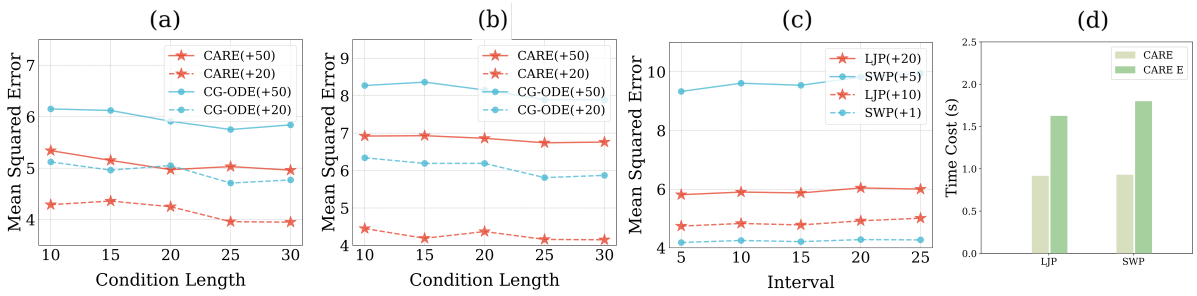


Figure 3.7: (a) and (b) show performance with respect to different condition and prediction lengths on the CylinderFlow and Airfoil datasets. (c) Examines the sensitivity of the interval for graph updating on the Lennard-Jones Potential (LJP) and 3-body Stillinger-Weber Potential (SWP) datasets. (d) Compares running time for dynamic graph updating versus full pairwise distance calculation on two particle-based datasets, demonstrating efficiency improvements.

Efficiency. Lastly, the efficiency of CARE's dynamic graph updating strategy was compared against a baseline method that calculates all pairwise distances for graph construction during each ODE step. The computational costs, as detailed in part (d) of Figure 3.7, reveal that CARE significantly reduces computational overhead, validating the model's practical applicability in real-world scenarios where computational resources are a constraint.

3.3.4 POEM: Hyperbolic ODE System for Mesh-based Simulations

Motivation

The motivation behind our proposed Hyperbolic ODE System (POEM) stems from critical challenges faced in the simulation of physical systems using traditional mesh-based methods and contemporary data-driven approaches. The field of physics simulations, which is crucial for advancements in areas ranging from mechanics to acoustics, has long relied on mesh-based finite element methods to describe complex interactions at various

mesh points. While these methods are proficient in allocating high-resolution resources to critical regions, they inherently produce complicated and irregular mesh structures that can be computationally intensive and less efficient for dynamic simulations.

Recent strides in deep learning have seen the introduction of graph neural networks (GNNs) to model physical interactions on these irregular meshes. These data-driven simulators, despite their innovative approach, primarily engage in learning from discretized data, which poses significant limitations in capturing continuous physical dynamics across mesh structures. This methodological constraint often leads to error accumulation and distribution shifts during long-term simulations, undermining the fidelity and reliability of the predictions.

Moreover, traditional simulation methods and recent GNN adaptations typically operate within a Euclidean framework, which may not naturally accommodate the hierarchical and multi-resolution nature of many physical systems. As physical systems like fluid dynamics and aerodynamics increasingly rely on multi-resolution meshes for efficient and accurate simulation, the need for a modeling approach that can inherently represent complex geometric and hierarchical structures becomes apparent.

Our motivation is further deepened by the inadequacies in the representation capabilities of Euclidean spaces for hierarchical data, which is a prevalent characteristic of advanced mesh-based models used in complex simulations. The hyperbolic space, with its unique geometric properties, offers a promising alternative for embedding such data with minimal distortion, a potential that has been largely untapped in physical simulations.

POEM is designed to address these challenges by leveraging the continuous nature of ordinary differential equations (ODEs) and the representational benefits of hyperbolic spaces. By embedding mesh points within a hyperbolic manifold, our model not only ensures a more natural representation of hierarchical structures but also enhances the interaction dynamics through a novel hybrid propagator mechanism. This mechanism

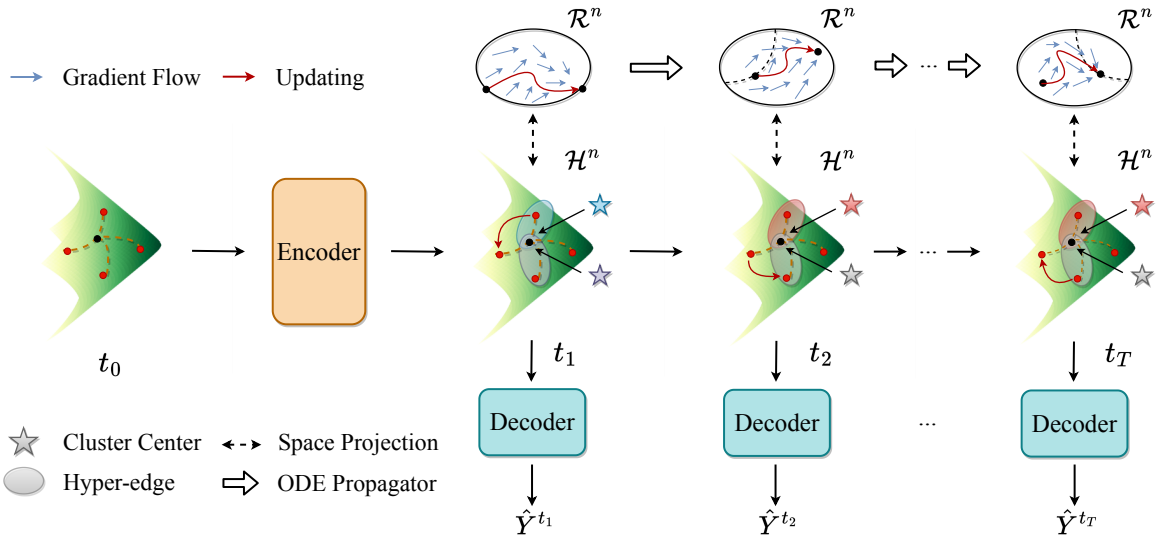


Figure 3.8: (a) The overview framework of the proposed POEM. Firstly, the encoder computes the latent initial states for mesh points in the hyperbolic space. Then the hybrid interacting propagator serves as the ODE function to drive the system to move forward, based on both the mesh-based graph and cluster-based hypergraph for efficient updating. Finally, a decoder outputs the predictions. (b) Details of our hybrid interacting propagator, which explores local and global interaction with neighboring nodes in the mesh graph and hyperedges in the hypergraph, respectively.

combines static mesh-based graphs with dynamic distance-based hypergraphs, facilitating both local and global interaction captures with lower computational overheads.

In summary, the motivation for developing POEM is to transcend the limitations of existing mesh-based and GNN methods by introducing a continuous, hyperbolic ODE-based framework that can more naturally and efficiently simulate dynamic physical systems, especially those characterized by complex, hierarchical mesh structures. Our approach aims to set a new standard in the simulation of physical systems, marrying the theoretical depth of continuous models with the practical efficacy of modern computational techniques.

Methodology

POEM introduces a novel architecture by leveraging hyperbolic spaces and ODE-based models for dynamic mesh-based systems. As shown in Figure 3.8 (a), the methodology consists of three primary components: an initial state encoder, a hybrid interacting propagator, and a decoder, each tailored to enhance the simulation's accuracy and efficiency.

Network Architecture POEM integrates three main components: an initial state encoder, an ODE-based propagator, and a decoder. Each component is tailored to process mesh-based simulations efficiently and effectively.

Initial State Encoder. The encoder's primary function is to transform the initial states of mesh points from the Euclidean to the hyperbolic space, thereby preparing the data for the subsequent ODE-based propagation. This transformation helps in capturing the hierarchical structure inherent among mesh points effectively. The notation $\mathbf{h}_i^{0,E}$ represents the hidden state of mesh point i in the Euclidean space:

$$\mathbf{h}_i^{0,E} = f_{\text{enc}}^E\left(\mathbf{x}_i\right), \tag{3.52}$$

Afterwards, each $\mathbf{h}_i^{0,E}$ is mapped to the hyperbolic space, denoted as $\mathbf{h}_i^{0,H}$, using the exponential map:

$$\mathbf{h}_i^0 = \mathbf{h}_i^{0,H} = \exp_{\mathbf{o}}([0, \mathbf{h}_i^{0,E}]). \tag{3.53}$$

Hybrid Interacting Propagator. This component of POEM is crucial for modeling the interactions among nodes in a mesh. Unlike traditional models that handle interactions in a purely Euclidean context, POEM utilizes a hyperbolic ODE system, allowing for a more nuanced interaction that considers both the static and dynamic relationships

among mesh points:

$$\frac{d\mathbf{h}_i^s}{ds} = \log_{\mathbf{h}_i^s}(f_{\mathbf{h}_i^s}(\mathcal{N}\left(\mathbf{h}_i^s\right))) \in \mathcal{T}_{\mathbf{h}_i^s}\mathcal{M}, \forall i. \tag{3.54}$$

where $\mathcal{N}(\mathbf{h}_i^s)$ encompasses both local ($\mathcal{N}^l$) and global ($\mathcal{N}^g$) neighborhood aggregations. These are defined as follows:

$$\begin{cases} \mathcal{N}^l\left(\mathbf{h}_i^s\right) = \sum\limits_{j \in N(i)} w(\mathbf{h}_i^s, \mathbf{h}_j^s) \log_{\mathbf{h}_i^s}\left(\mathbf{h}_j^s\right) \\ \mathcal{N}^g\left(\mathbf{h}_i^s\right) = \sum\limits_{k=1}^{K} w(\mathbf{h}_i^s, \mathbf{e}_k^s) \log_{\mathbf{h}_i^s}\left(\mathbf{e}_k^s\right), \end{cases} \tag{3.55}$$

where $w(\cdot, \cdot)$ is a weight function modeled by a shared MLP, calculating normalized weights for both local and hyperedge interactions.

The evolving nature of mesh point interactions, especially in dynamic settings, is captured by periodically updating the interaction hypergraph based on online clustering, thus ensuring that all significant interactions are accounted for efficiently.

Decoder. The decoder uses the trajectory information, predicted by the propagator, to generate final outputs:

$$\hat{\mathbf{y}}_i^t = f_{\text{dec}}(\mathbf{h}_i^s), \tag{3.56}$$

Model Optimization To ensure the practical implementation of our ODE model within the constraints of hyperbolic space, we design an update strategy that approximates the continuous dynamics in a computationally feasible manner. Standard ODE solvers typically operate in Euclidean spaces; therefore, we approximate the dynamics within small time intervals to efficiently train our model in the tangent space at each mesh point:

$$\frac{d\mathbf{h}_i^{s,E}}{ds} = \log_{\mathbf{h}_i^t}(f_{\mathbf{h}_i^t}(\exp_{\mathbf{h}_i^t}(\mathcal{N}^l(\mathbf{h}_i^s) + \mathcal{N}^g(\mathbf{h}_i^s)))), \tag{3.57}$$

where $\mathbf{h}_i^{s,E} = \log_{\mathbf{h}_i^t}(\mathbf{h}_i^s)$. This equation defines how we map hyperbolic embeddings back to the tangent space at each interval, allowing us to leverage traditional numerical methods for ODEs.

We discretize the total simulation time $[0, T]$ into uniform intervals $[0, t_1], \ldots, [t_{L-1}, T]$. At each interval, the hyperbolic embeddings from the initial state $\mathbf{h}_i^t$ are transformed into their Euclidean tangent space representations $\mathbf{h}_i^{s,E}$, and the ODE is solved in this space. The solution is then mapped back to the hyperbolic space, ensuring consistency with the model's geometric constraints.

To facilitate this transformation between tangent spaces across intervals, we use the combination $\log_{\mathbf{h}_i^{t_l}} \circ \exp_{\mathbf{h}_i^{t_{l-1}}}$, which is differentiable due to the diffeomorphic nature of the logarithmic and exponential maps. This approach ensures that the forward integration process across different intervals remains smooth and differentiable, allowing the entire model to be optimized using standard ODE solvers.

$$f_{\mathbf{h}_i^t} = \exp_{\mathbf{h}_i^t} \circ f_{\text{pro}} \circ \log_{\mathbf{h}_i^t}, \tag{3.58}$$

where f_{pro} is an MLP designed for feature transformation, simplifying the interaction calculations by maintaining operations within the Euclidean space of the tangent bundle.

This strategy not only allows for efficient training but also ensures that our model adheres closely to the dynamics dictated by the underlying physical principles, thus maintaining the accuracy and robustness of the simulation over time.

Theoretical Analysis The theoretical foundation of POEM ensures that the simulations are not only efficient but also adhere closely to the dynamics dictated by the un-

derlying physical principles. The hyperbolic manifold and the ODE framework together facilitate a robust simulation environment that can handle complex, multi-resolution mesh structures naturally and efficiently.

Assumption 12. The mesh representations are Lipschitz-continuous and bounded in Lorentz norm, i.e., we have constants $C_1, C_2 > 0$, s.t. $\forall i,\ s \in [t, t + \Delta t]$,

$$\|\mathbf{h}_i^s - \mathbf{h}_i^t\|_{L^2} \leq C_1(s - t),\ |\|\mathbf{h}_i^t\|_{\mathcal{L}}| \leq C_2. \tag{3.59}$$

Lemma 6. For any given $\varepsilon > 0$, we can find $\Delta t > 0$, s.t. the difference between solutions to the ODE system

$$\frac{d\mathbf{h}_i^{s,E}}{ds} = \log_{\mathbf{h}_i^t} f_{\mathbf{h}_i^s}(\exp_{\mathbf{h}_i^s}(\mathcal{N}^l(\mathbf{h}_i^s) + \mathcal{N}^g(\mathbf{h}_i^s))), \tag{3.60}$$

and Eqn. 3.57 is bounded by $\varepsilon \cdot \Delta t,\ \forall s \in [t, t + \Delta t]$.

Corollary 13. Given the initial state $\mathbf{h}^t$, we claim that for any given $\varepsilon > 0$, we can find Δt, s.t. the difference between solutions to the model Eqn. 3.54 and our approximation Eqn. 3.57 is bounded by $C\varepsilon \cdot \Delta t$ on interval $[t, t + \Delta t]$ for some constant C.

The proofs of these theoretical assertions are detailed in the Appendix A, providing a rigorous mathematical underpinning to the proposed methodology. This theoretical framework not only supports the operational efficacy of POEM but also reassures its adaptability and scalability to various physical simulation scenarios, particularly those involving complex and hierarchical mesh structures.

Experiment Results

Our methodology is rigorously tested on four distinct datasets from various physical domains: *CylinderFlow*, *Airfoil*, *DeformingPlate*, and *FlagSimple*. Each dataset embod-

ies a different aspect of physical simulations—ranging from fluid dynamics to structural mechanics and cloth modeling—ensuring a comprehensive evaluation of our model's capabilities across diverse settings.

Experimental Setup The datasets utilized offer a challenging array of dynamics:

- *CylinderFlow* involves predicting the velocity fields and pressure around a cylinder within an incompressible flow.
- *Airfoil* examines the aerodynamics at an airfoil cross-section with predictions focusing on velocity, pressure, and density fields.
- *DeformingPlate* deals with the structural dynamics of a deforming plate.
- *FlagSimple* captures the movement of a flag in wind, focusing on acceleration predictions.

Each dataset comprises 1000 training samples, with 100 samples each for validation and testing. The complexity of the mesh interactions, combined with the hyperbolic space modeling, poses a unique set of challenges addressed by our model. We employ a cluster number of 15 for constructing dynamic hypergraphs and set the interval length to 10, details of which are discussed in the Appendix.

Quantitative Analysis The effectiveness of our model is quantitatively validated against several state-of-the-art models including GNS, MeshGraphNet, and TIE, particularly in long-term prediction scenarios. Our model demonstrates substantial improvements in accuracy across all datasets, significantly reducing the average error in long-term predictions compared to the baselines. The detailed results are presented in Table 3.8, showing a marked decrease in average RMSE across different physical systems.

Table 3.8: The average error of all the time steps on four systems, with unit of $\times 10^{-3}$. We compare our proposed method with GNS, MeshGraphNet and TIE on four benchmark physics simulation datasets. Our model significantly outperforms all the baselines on four datasets for long-term predictions.

Method	**CylinderFlow**			**Airfoil**				**DeformingPlate**			**FlagSimple**	
	v_x	v_y	p	v_x	v_y	p	d	v_x	v_y	s	a_x	a_y
LSTM [49]	78	912	524	18569	10046	1084	987	12.4	18.7	52	184	496
GNS[107]	82	981	519	20718	11034	1234	1011	12.6	19.6	57	201	513
STGCN[108]	46	525	285	15792	9564	989	864	8.1	14.1	31	136	313
MeshGraphNet[109]	38	521	274	13246	7623	974	528	8.6	12.8	29	122	255
MP-NODE[110]	15	208	105	6514	4523	455	396	7.4	11.6	21	81	124
TIE[111]	11	215	92	5231	4016	430	377	7.5	13.2	20	78	129
POEM (Ours)	**3.6**	**96**	**48**	**2798**	**2006**	**272**	**169**	**7.1**	**10.4**	**17**	**49**	**103**

Visualization of Results We provide detailed visualizations of the velocity fields for both *CylinderFlow* and *Airfoil*. These visualizations not only highlight the precision of our model in capturing complex fluid dynamics but also its ability to maintain accuracy over extended simulation periods. This capability is particularly noted in the stable prediction of long-term dynamics, which is a notable challenge for existing simulation methods. Figure 3.9 illustrates these comparisons.

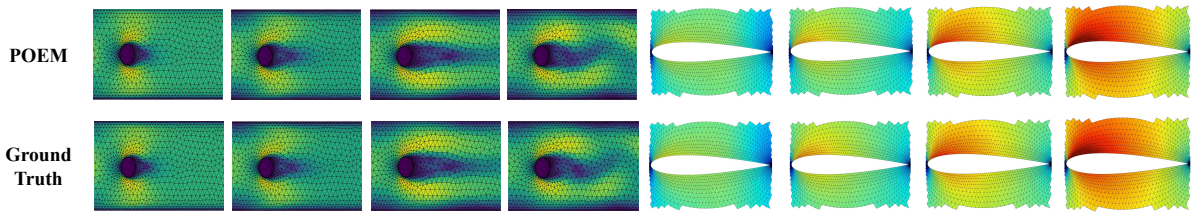


Figure 3.9: The visualization of the velocity field for (left) water flow around a cylinder obstacle, and (right) air around the cross-section of an aircraft wing. We show that our proposed POEM can accurately make long-term predictions on varying system variables.

Statistical Performance Comparison Further statistical analysis reveals that our model outperforms traditional and recent GNN-based methods by a significant margin. For instance, in the *CylinderFlow* dataset, our model achieves a 56.81% reduction in RMSE compared to the best-performing baseline. Similar trends are observed across other datasets, reinforcing the superiority of our hyperbolic ODE-based approach in handling complex mesh dynamics and long-term dependencies. Figure 3.10 displays these comparisons graphically.

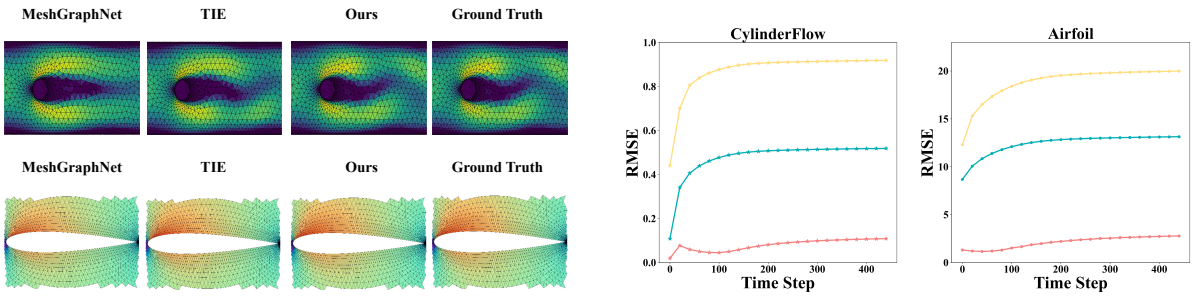


Figure 3.10: Left: The simulation results of MeshGraphNet, TIE, and our model at the time step 400 on CylinderFlow and Airfoil compared to the ground truth. Our model can make the most accurate predictions in both cases. Right: Averaged RMSE (v_x) over all mesh points with respect to different time steps on CylinderFlow and Airfoil.

Trade-off between Effectiveness and Efficiency As highlighted in our theoretical analysis, optimizing the interval length (i.e., T/L) is crucial for balancing computational cost and model accuracy. We evaluate performance across different interval lengths to identify the optimal setting, which is depicted in Figure 3.11. Our findings indicate that performance improves with decreasing interval length until it reaches a saturation point, beyond which further reductions yield diminishing returns. This trade-off is critical in practical applications where both precision and efficiency are valued.

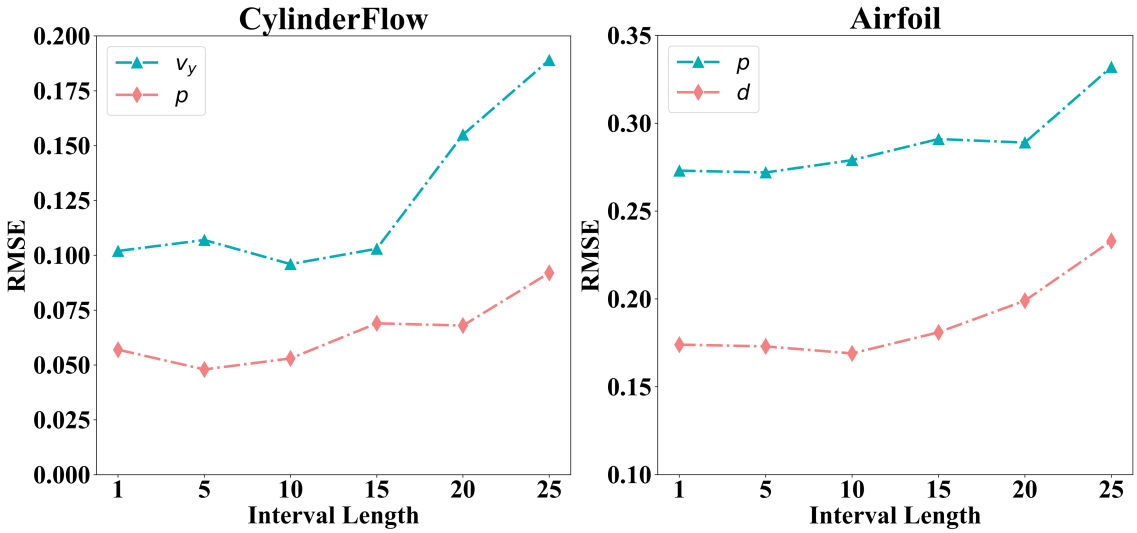


Figure 3.11: Performance with respect to different interval lengths on CylinderFlow and Airfoil. The graph demonstrates the trade-off between simulation accuracy and computational efficiency.

Ablation Study To validate the effectiveness of the individual components of our model, we conduct an ablation study. We compare our full model against three variants: (i) *POEM w.o. HB*, which does not use hyperbolic embeddings; (ii) *POEM w.o. M*, which excludes the mesh-based graph component; and (iii) *POEM w.o. HG*, which omits the dynamic hypergraph component. The results, presented in Table 3.9, highlight the significant impact of each component on the model's performance, particularly the crucial role of hyperbolic embeddings in managing the complexity and hierarchy of the mesh structures.

Table 3.9: Ablation studies on three key components of our method. POEM w.o. HB learns mesh representations in the Euclidean space; POEM w.o. M removes the usage of the mesh-based graph; POEM w.o. HG removes the usage of the dynamic hypergraph.

Method	CylinderFlow			Airfoil				DeformingPlate			FlagSimple	
Variable	v_x	v_y	p	v_x	v_y	p	d	v_x	v_y	s	a_x	a_y
POEM w.o. M	18.74	278	113	8976	5655	607	414	11.3	19.1	48	231	584
POEM w.o. HB	7.9	177	75	5679	4428	489	354	7.4	10.9	21	164	289
POEM w.o. HG	4.1	105	66	2954	2480	294	187	7.2	10.8	22	54	108
POEM	**3.6**	**96**	**48**	**2798**	**2006**	**272**	**169**	**7.1**	**10.4**	**17**	**49**	**103**

3.3.5 GraphSDE: Modeling Stochastic Dynamics for Interacting Systems

Motivation

In the real-world, interacting dynamical systems, such as those in autonomous driving, climate modeling, and stock market forecasting, exhibit inherent uncertainties and complexities that deterministic models struggle to capture accurately. This limitation stems from the deterministic nature of traditional modeling approaches, which often assume that future trajectories follow a predictable path. However, many real-world scenarios inherently contain multiple potential outcomes due to their stochastic nature, such as weather patterns and financial markets. Recognizing these challenges, we propose a novel method, GraphSDE, to model the stochastic dynamics of such systems more effectively.

Our motivation for developing GraphSDE arises from the critical need to handle the inherent randomness present in every small time interval of real-world systems, which is often overlooked by existing deterministic or discretely stochastic approaches. These methods typically predict a single trajectory or a set of discrete outcomes, failing to account for the continuous stochastic nature of the underlying processes. By integrating continuous stochastic processes within a graph-based framework, GraphSDE aims to pro-

vide a more robust and realistic modeling of dynamical systems by generating a spectrum of possible outcomes. This approach not only enhances the accuracy of predictions in systems with inherent uncertainties but also empowers decision-makers to evaluate risks and make informed decisions based on a range of potential scenarios.

Furthermore, the introduction of GraphSDE addresses the challenge of modeling complex interactions within these systems, where the dynamics are driven by intricate interdependencies among individual components. By employing a conditional neural SDE that incorporates drift and diffusion processes, our method captures both the individual behaviors and the collective dynamics influenced by stochastic interactions. This holistic approach ensures that GraphSDE can effectively represent the detailed interactions and randomness, providing a significant advancement over traditional methods in capturing the true complexity of interacting dynamical systems.

Methodology

GraphSDE encompasses an encoder for initial state estimation, a conditional graph SDE for modeling dynamic systems with inherent stochasticity, and a decoder for trajectory prediction, optimized using a variational inference framework.

Spatio-Temporal Graph Encoder The encoder initializes node states by extracting spatio-temporal features from historical data, leveraging a message passing mechanism to update node representations:

$$\mathbf{h}_i^{t,(l)} = f^{mp}([\mathbf{h}_i^{t,(l-1)}, \mathbf{h}_i^{t-1,(l-1)}, \mathrm{AGG}(\{\mathbf{h}_j^{t,(l-1)}, j \in \mathcal{S}^{(i)}\})]), \tag{3.61}$$

where $\mathrm{AGG}(\cdot)$ is an aggregation function such as sum or average. The final node representation $\mathbf{u}_i$ is obtained through attention mechanisms over all time steps, allowing the

estimation of Gaussian distribution parameters for the variational inference:

$$q(\mathbf{z}_i^0|\mathbf{X}^{ob}, G) = \mathcal{N}(f^{enc,m}(\mathbf{u}_i), f^{enc,v}(\mathbf{u}_i)), \quad \mathbf{z}_i^0 \sim q(\mathbf{z}_i^0|\mathbf{X}^{ob}, G). \tag{3.62}$$

Conditional Graph SDE To address the limitations of Neural ODEs, particularly their deterministic nature, we introduce a Conditional Graph SDE model to incorporate continuous randomness into dynamical systems modeling. This approach enables us to capture complex interacting dynamics by combining individual trends, interactive influences, and global system states in a hierarchical feature space. This complexity is essential for accurately representing real-world systems that exhibit inherent stochastic behavior.

In the implementation, we first sample hidden features $\mathbf{z}_i^0$ for each node and generate hierarchical features by aggregating local and global information:

$$\mathbf{n}_i^0 = \mathrm{AGG}^l(\{\mathbf{z}_j^0, j \in \mathcal{N}(i)\}), \mathbf{g}_i^0 = \mathrm{AGG}^g(\{\mathbf{z}_j^0, j \in V\}), \tag{3.63}$$

where $\mathrm{AGG}^l(\cdot)$ and $\mathrm{AGG}^g(\cdot)$ aggregate features from local neighbors and the entire graph, respectively. The local and global features are critical for capturing the dynamics of the system as they provide insights from different perspectives of the graph structure.

These features are used to initialize the state of each node in the graph SDE, which is then defined as:

$$d\mathbf{h}_i^t = \phi_c^{dr}\left(\hat{\mathbf{h}}_i^t, t\right) dt + \phi_c^{di}\left(\hat{\mathbf{h}}_i^t, t\right) d\mathbf{W}^t, \tag{3.64}$$

with $\hat{\mathbf{h}}_i^t$ incorporating the concatenated features of local interactions and global state

influences:

$$\hat{\mathbf{h}}_i^t = \begin{bmatrix} \mathbf{z}_i^t \\ \mathrm{AGG}^l(\{\mathbf{n}_i^t, j \in \mathcal{N}(i)\}) \\ \mathrm{AGG}^g(\{\mathbf{g}_i^t, j \in V\}) \end{bmatrix}, \tag{3.65}$$

where ϕ_c^{dr} and ϕ_c^{di} represent the drift and diffusion terms, respectively, and $\mathbf{W}^t$ is a multi-dimensional Wiener process, adding stochasticity to the system's evolution.

Stability Analysis. To ensure the robustness and reliability of our SDE model, we analyze its stability under perturbations. We assume the linear aggregation operators, $\mathrm{AGG}^l(\cdot)$ and $\mathrm{AGG}^g(\cdot)$, allow us to represent the drift and diffusion functions in a linearized form:

$$d\varepsilon^t = f_\Delta(\varepsilon^t, t)dt + g_\Delta(\varepsilon^t, t)d\mathbf{W}^t, \tag{3.66}$$

where

$$f_\Delta(\varepsilon^t, t) = \mathbf{M}^{t,dr}\varepsilon^t, \quad g_\Delta(\varepsilon^t, t) = \mathbf{M}^{t,di}\varepsilon^t, \tag{3.67}$$

and $\mathbf{M}^t = \mathbf{W}^t\mathbf{A}^t$ represents the message passing matrix. The stability of our model is supported by the following lemma:

Lemma 7. Suppose $\exists\lambda_{\max}, \lambda_{\min} > 0$, s.t $\forall t > 0$, the eigenvalues for $\mathbf{M}^t$ satisfy:

$$0 < \lambda_{\min} \leq |\lambda_1^t| \leq \cdots \leq |\lambda_{3dN}^t| \leq \lambda_{\max}. \tag{3.68}$$

Then we claim that the following inequality is a sufficient but not necessary condition to guarantee that the solution $\varepsilon^t \equiv \mathbf{0}$ to Eqn. 3.66 is almost surely exponentially stable:

$$2 \leq \lambda_{\min} \leq \lambda_{\max} \leq \sqrt{2\lambda_{\min}^2 + 1} - 1. \tag{3.69}$$

The proof is provided in Appendix A. This lemma is foundational for our stabil-

ity claims, demonstrating robustness against small perturbations and ensuring reliable predictions under dynamic conditions.

Decoder and Optimization Decoder. Our decoder function maps the sampled latent embeddings $\mathbf{z}_i^t$ from the GraphSDE model to trajectory predictions, using a fully connected network:

$$\hat{\mathbf{x}}_i^t = f^{dec}(\mathbf{z}_i^t), \tag{3.70}$$

where $f^{dec}(\cdot)$ represents the decoding network. This setup allows for the generation of diverse trajectory predictions by sampling multiple outcomes from the latent distribution, reflecting the probabilistic nature of future states in the dynamical system.

Optimization. The model is optimized by maximizing the evidence lower bound (ELBO), balancing the likelihood of the generated trajectories with the complexity of the model. The ELBO formulation is as follows:

$$\mathcal{L} = \mathbb{E}_{\mathbf{Z}^0 \sim \prod_{i=1}^N q\left(\mathbf{z}_i^0 | \mathbf{X}^{ob}, \mathcal{C}\right)} \left[\log p(\mathbf{X}^{T'+1:T} | \mathbf{Z}^0, \mathcal{C})\right] - \mathrm{KL}\left[\prod_{i=1}^N q(\mathbf{z}_i^0 | G^{1:T_{obs}}, \mathcal{C}) \| p\left(\mathbf{Z}^0 | \mathcal{C}\right)\right], \tag{3.71}$$

where $\mathcal{C}$ represents the class labels that condition the model.

Robustness. Ensuring robustness in stochastic modeling is critical, especially when dealing with real-world dynamical systems that are subject to various perturbations and uncertainties. We define the robustness of our model as follows:

Definition 19. A model f, operating on an input $\mathbf{x}$ and outputting a continuous random variable, is considered r-robust for a radius $r \in \mathbb{R}^+$ against a perturbation δ, if $\mathbb{P}(\|f(\mathbf{x}+\delta) - f(\mathbf{x})\| \leq r) > \mathbb{P}(\|f(\mathbf{x}+\delta) - f(\mathbf{x})\| > r)$.

Under this framework, we propose the following theorem, leveraging the stability conditions established earlier to demonstrate enhanced robustness margins in our SDE

model compared to traditional ODE models:

Theorem 14. Under the assumption of Lemma 7, let $\mathcal{R}^r_{\mathcal{X}}(t)$ denote the robustness margin for the SDE model as per Eqn. 3.64, solved via the autoencoder framework in Eqn. 3.71. Assume $\tilde{\mathcal{R}}^r_{\mathcal{X}}(t)$ represents the robustness margin for the following ODE model:

$$d\mathbf{h}_i^t = \phi_c^{dr}\left(\hat{\mathbf{h}}_i^t, t\right) dt, \tag{3.72}$$

which removes the diffusion term $\int \phi^{di}\left(\mathbf{z}^s, s\right) \mathrm{d}\mathbf{W}^s$ solved using Eqn. 3.71. We further assume there exist positive constants $c_1,\ c_2$ such that:

$$c_1\|\mathbf{z}_1 - \mathbf{z}_2\| \leq g^t(\mathbf{z}_1 - \mathbf{z}_2) \leq c_2\|\mathbf{z}_1 - \mathbf{z}_2\|. \tag{3.73}$$

Then, we can have:

$$\mathcal{R}^r_{\mathcal{X}}(t) \geq \tilde{\mathcal{R}}^r_{\mathcal{X}}(t)\ \forall r > 0 \text{ and } t = T' + 1, \ldots, T. \tag{3.74}$$

The proof, detailed in Appendix A, showcases the model's capability to maintain reliable predictions despite inherent uncertainties and variabilities, underscoring the significance of incorporating stochastic elements into the dynamical systems modeling.

Experiment Results

Datasets and Settings. We assess GraphSDE on four dynamic system datasets: **COVID-19**, **Radar Map**, **Stock**, and **Social Network**. These datasets encapsulate a range of dynamics from pandemic progression to financial fluctuations and social interactions. Detailed descriptions are available in Appendix B.

Baselines. GraphSDE is benchmarked against several state-of-the-art models includ-

ing LSTM, GRU, NODE, and others, detailed in Appendix B. We adopt the Best-of-N strategy and use MAE, RMSE, and MAPE for evaluation, further elaborated in Appendix B.

Table 3.10: The compared results on COVID-19 over 7-Day, 14-Day, and 21-Day periods. **Bold** numbers highlight the best performance whereas underline numbers highligh the second best performance.

Methods	7-Day Average			14-Day Average			21-Day Average		
	MAE	RMSE	MAPE(%)	MAE	RMSE	MAPE(%)	MAE	RMSE	MAPE(%)
LSTM	30.1±0.3	39.0±0.6	15.3±1.3	29.8±0.2	40.3±0.3	26.2±1.5	31.6±0.1	45.6±0.2	34.8±1.5
GRU	24.7±0.5	34.9±0.9	8.19±0.35	26.5±0.3	41.0±0.4	11.6±0.1	30.0±0.2	49.4±0.3	14.4±0.2
NODE	29.3±0.7	39.3±1.1	10.6±0.5	30.5±0.8	40.8±1.5	18.4±0.7	32.1±0.8	44.1±1.5	25.2±0.8
HBNODE	30.7±0.8	41.3±1.4	10.3±0.5	30.0±0.7	39.5±1.4	18.7±0.6	32.0±0.7	46.9±1.5	22.5±0.6
MPNODE	17.7±0.2	26.1±0.3	3.54±0.26	19.4±0.1	28.8±0.2	6.07±0.47	21.5±0.1	33.7±0.1	8.32±0.65
CG-ODE	16.3±0.3	24.1±0.2	2.48±0.32	18.2±0.2	29.8±0.2	4.26±0.43	22.3±0.1	33.0±0.3	6.26±0.54
HOPE	16.1±0.3	23.9±0.3	2.31±0.22	18.3±0.3	28.3±0.3	4.15±0.46	21.7±0.3	32.4±0.2	6.12±0.64
GraphSDE	**15.8±0.2**	**23.8±0.3**	**2.26±0.21**	**17.8±0.3**	**27.5±0.2**	**3.90±0.36**	**19.4±0.5**	**29.5±0.1**	**6.04±0.51**

Table 3.11: The compared results on Radar Map over cumulative 3-Spans and 5-Spans periods, each span representing a 6-minute interval. MAE and RMSE values are presented in units of 10^{-3}.

Methods	3-Spans Average		5-Spans Average	
	MAE	RMSE	MAE	RMSE
LSTM	46.9±0.1	76.7±0.1	58.1±0.2	88.3±0.1
GRU	48.2±0.2	80.2±0.2	60.0±0.1	90.6±0.2
NODE	46.7±0.1	76.2±0.1	56.8±0.2	86.2±0.2
HBNODE	47.1±0.3	77.1±0.2	57.2±0.3	87.1±0.1
MPNODE	52.1±0.2	82.3±0.1	61.7±0.2	93.1±0.3
CG-ODE	53.3±0.2	82.5±0.2	64.1±0.3	95.4±0.1
HOPE	52.8±0.2	81.8±0.3	62.8±0.2	93.7±0.3
GraphSDE	**45.5±0.3**	**70.3±0.1**	**56.0±0.2**	**85.5±0.2**

Performance Comparison. Our evaluations, summarized in Tables 3.10 to 3.11, show that GraphSDE consistently outperforms the baselines across all datasets. Notably, on the Stock dataset, it reduces prediction errors significantly compared to the best baseline, MPNODE. The introduction of stochastic elements and hierarchical interactions in GraphSDE effectively captures the complex dynamics and randomness in these systems, enhancing predictive accuracy.

Visualization and Insights. We visualize the results of different approaches on

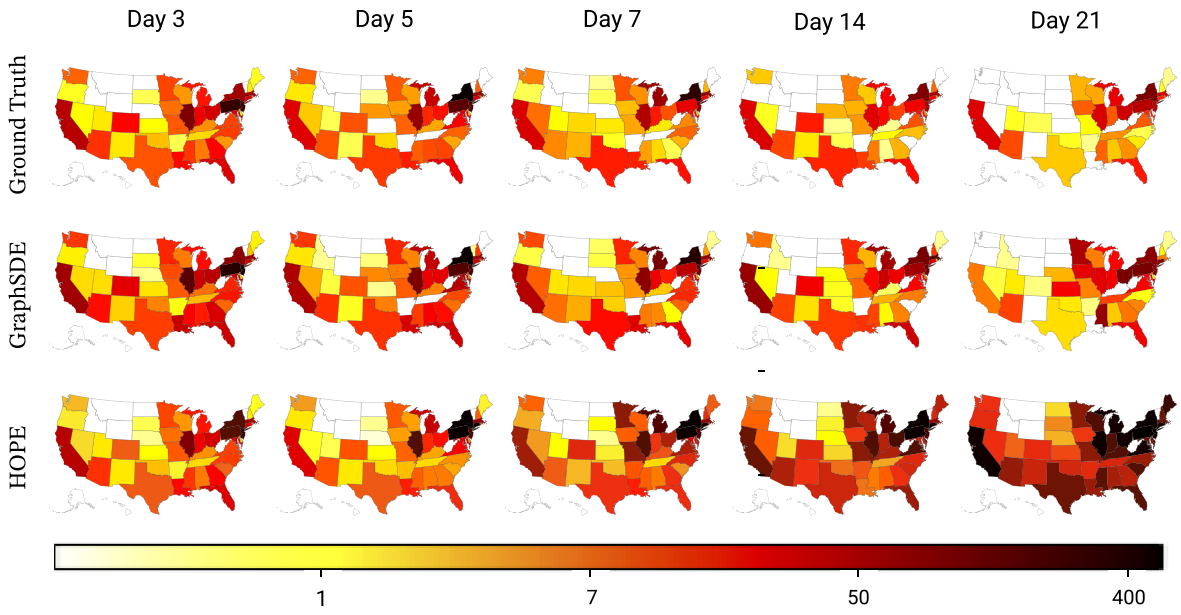


Figure 3.12: Comparative results of HOPE and our GraphSDE on COVID-19 fatalities in the US over three weeks, using a logarithmic scale for better visibility.

the COVID-19 and Radar Map datasets. The predictive capability of GraphSDE is illustrated as follows:

To illustrate the capability of capturing the dynamic nature of epidemic spread on the COVID-19 dataset, we map the daily fatalities in the United States to intuitively depict the progression or mitigation trends of the pandemic.

The Radar Map results show that our GraphSDE can generate accurate predictions in areas with sudden shifts over time intervals, demonstrating superior performance in long-term prediction due to our continuous modeling of stochastic dynamical systems with reduced error accumulation.

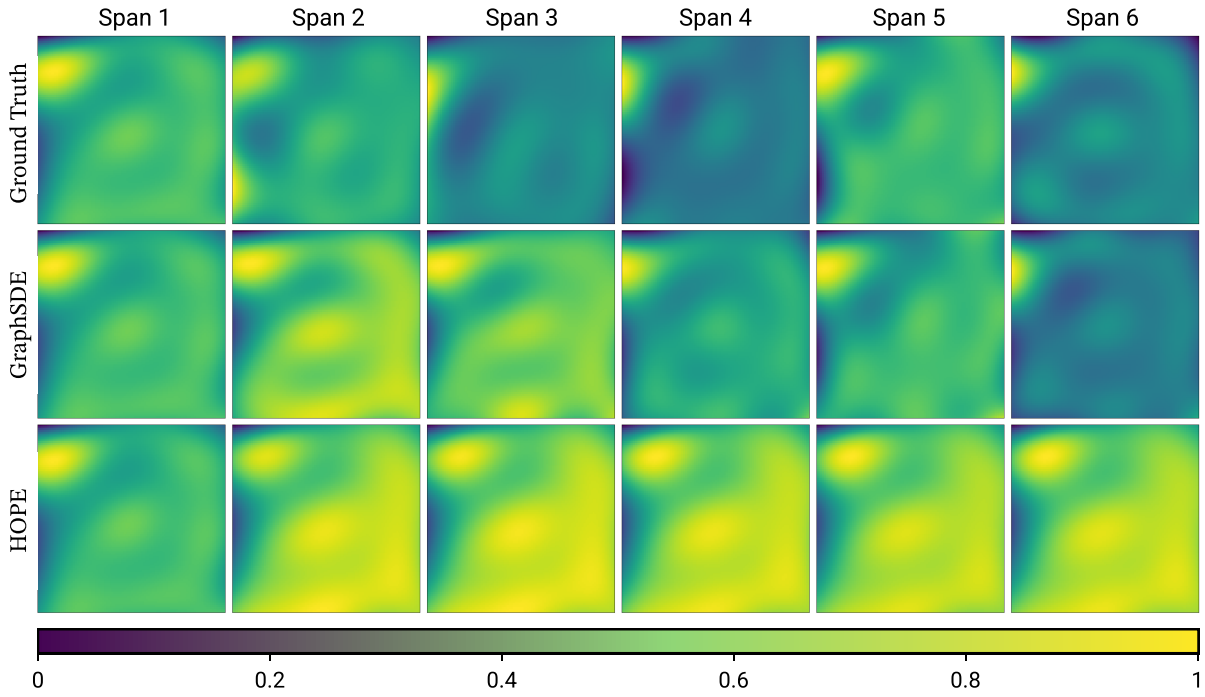


Figure 3.13: Comparison of HOPE and our GraphSDE on Radar Map reflectivity, with RBF Interpolation for reconstructing location data from 30 observation points.

Table 3.12: Comparisons between our GraphSDE and its variants on Stock data.

Methods	10-day Average			20-day Average			40-day Average		
	MAE	RMSE	MAPE(%)	MAE	RMSE	MAPE(%)	MAE	RMSE	MAPE(%)
GraphSDE w/o G	31.4±0.2	94.5±0.4	**17.4±0.2**	34.1±0.2	94.5±0.5	18.5±0.3	37.1±0.2	99.2±0.4	21.2±0.3
GraphSDE w/o L	35.1±0.3	120.4±0.5	22.1±0.3	38.2±0.3	103.6±0.4	22.4±0.2	39.7±0.1	108.8±0.4	24.2±0.2
GraphSDE w/o D	33.7±0.4	105.2±0.4	19.2±0.2	36.8±0.4	99.8±0.4	21.6±0.3	38.2±0.2	102.1±0.3	20.8±0.2
GraphSDE w/o C	31.9±0.2	97.4±0.5	18.0±0.3	35.2±0.2	98.3±0.4	20.4±0.1	37.7±0.2	100.8±0.4	21.1±0.3
GraphSDE	**31.0±0.2**	**92.5±0.5**	17.6±0.2	**32.5±0.1**	**90.0±0.5**	**17.8±0.1**	**35.0±0.1**	**94.1±0.4**	**18.4±0.1**

Ablation Study. We conduct ablation studies on the Stock dataset to evaluate the different contributions of each component in GraphSDE. The results, presented in Table 3.12, affirm the significance of both local and global messages, the diffusion components, and the mode-specific SDE configurations in capturing the nuanced stochastic dynamics

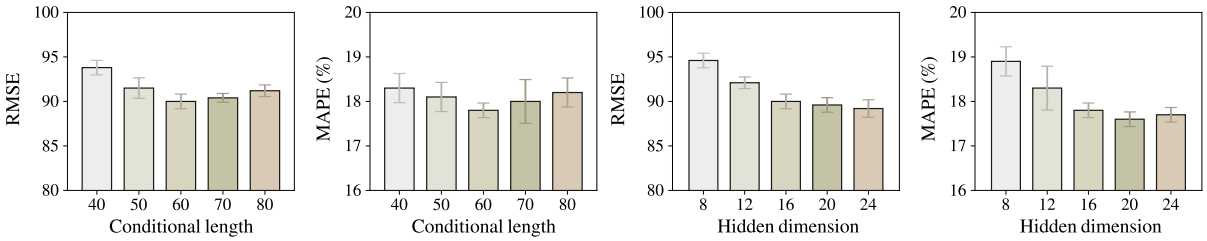


Figure 3.14: Sensitivity analysis on a 20-day forecast task for Stock, showing accuracy and 80% confidence intervals.

of the system.

Sensitivity Analysis. We examine how key hyperparameters, condition sequence length T_{cond} and hidden dimension d, impact the performance of GraphSDE on the Stock dataset:

Increasing condition length typically reduces predictive errors before reaching saturation, indicating that more historical trajectories can enhance performance. However, overly long historical trajectories might degrade performance due to potential overfitting. Increasing the hidden dimension generally improves performance due to enhanced model representation capacity, stabilizing once dimensions exceed 16.

Chapter 4

Training strategy: Channel Independent (CI) versus Channel Dependent (CD)

4.1 Introduction and Related Work

Transformer-based architectures [55] have achieved remarkable success in various time series modeling tasks, including forecasting, classification, and regression. Notable examples include Informer [57], Autoformer [58], and FedFormer [59], among others. However, as studied in [60], when the input sequence length increases, the performance of transformers does not improve proportionally, suggesting potential overfitting during training. This observation has prompted the exploration of alternative architectures that can effectively capture long-range dependencies while mitigating overfitting.

One such alternative is the Dlinear architecture proposed in [60], which employs a simple multi-layer perceptron (MLP) architecture. Although Dlinear may not initially outperform transformers, its prediction error decreases as the input sequence length in-

creases, indicating its ability to learn from more input data. This contrasting behavior has led to a rethinking of the transformer architecture, leading to the development of iTransformer [112] and Crossformer [113].

The iTransformer [112] introduces a novel channel-wise embedding strategy, where each channel is embedded separately before being combined through cross-channel interactions. This approach aims to capture the intrinsic dynamics of individual channels while maintaining the ability to model cross-channel dependencies. The Crossformer [113], on the other hand, proposes a cross-channel attention mechanism, allowing efficient capture of channel-wise and cross-channel dependencies.

Furthermore, the work of [114] revisits the channel-independent (CI) and channel-dependent (CD) strategies for multivariate time series forecasting. The authors introduce a distribution difference metric to quantify the divergence between the distributions of the training and test set. Through visualizations and empirical analysis, they demonstrate that for most real-world datasets, there exists a significant distributional shift between the training and test sets. This observation provides insights into the potential reasons behind the superior performance of the CI strategy over the CD strategy in certain scenarios, as the CI strategy may be more robust to distributional shifts.

The authors of [114] also investigate the trade-off between the capacity and robustness of the CI and CD strategies. They argue that while the CD strategy has a higher capacity to fit the training data, it may suffer from reduced robustness when faced with distributional shifts, leading to inferior performance on the test set. In contrast, the CI strategy, despite having lower capacity, exhibits greater robustness to distributional shifts, potentially explaining its better generalization in certain cases.

In this chapter, we build on these recent developments and insights, aiming to provide a comprehensive theoretical analysis of CI and CD strategies, quantifying the distributional shift between training and test sets, and exploring the trade-off between capacity

and robustness in these approaches. Our analysis will contribute to a deeper understanding of the underlying dynamics and guide the development of more effective and robust time series forecasting models.

4.2 Preliminaries

4.2.1 Overview of the Transformer

Building upon the remarkable success of Transformer-based architectures in various time series modeling tasks, as highlighted in the introduction, this subsection aims to delve deeper into the core components that make the Transformer model particularly effective. Central to the Transformer's ability to handle sequential data is the attention mechanism, which allows the model to dynamically focus on relevant parts of the input sequence when generating the output.

The basic unit of computation in a Transformer is the attention mechanism [55], specifically designed to compute a weighted sum of the values, the weights themselves being determined by a compatibility function between each value and a given query. The mathematical formulation of the attention function is as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{4.1}$$

Here, Q, K, and V represent the matrices of queries, keys, and values, respectively, each derived from the input data. d_k is the dimension of the keys, and scaling by $\sqrt{d_k}$ is done to prevent the dot products from growing too large in magnitude, which could lead to computational difficulties during training.

The queries, keys, and values are computed as linear transformations of the input

embeddings:

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V \tag{4.2}$$

where X denotes the input embeddings, and W^Q, W^K, and W^V are the weight matrices that are learned during training.

Extending the attention mechanism to multi-head attention allows the model to explore different subspace representations of the input sequence concurrently, enabling it to capture various types of relationships between the elements of the input sequence:

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \\ \text{where} \quad \text{head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{4.3}$$

In this configuration, W_i^Q, W_i^K, and W_i^V are the projection matrices for the i-th attention head, transforming the input into different subspaces for queries, keys, and values, respectively. W^O is another learned matrix that combines the outputs of all the heads of attention into a single output.

This framework of multi-head attention enables the Transformer to capture various aspects of the data's structure, accommodating complex dependencies across different positions in the sequence. This capability is particularly advantageous in multivariate time series forecasting, where understanding intricate patterns across multiple variables is crucial.

Position-wise Feed-Forward Networks Each layer of the Transformer includes a position-wise feedforward network (FFN), which applies the same two-layer neural network to each position separately and identically. This consists of two fully connected layers with a ReLU activation function in between, applied independently to each posi-

tion in the sequence:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{4.4}$$

where W_1, b_1, W_2, and b_2 are learnable parameters. The FFN allows the Transformer to apply a further non-linear transformation to the output of the attention mechanism, enhancing the model's ability to represent complex functions.

Positional Encoding Since the Transformer architecture lacks recurrence or convolution, positional encodings are added to the input embeddings to incorporate information about the order of the sequence. This is necessary because the self-attention mechanism alone does not have a notion of the relative positions of the input elements. The positional encodings are defined as:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right), \quad PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \tag{4.5}$$

where pos is the position in the sequence, and i is the dimension. These positional encodings are added to the input embeddings to provide the model with some notion of token position within the sequence.

The positional encodings are crucial for enabling the self-attention layers to interpret the sequences effectively, considering not only the content but also the order of the data points. The added positional information allows the attention mechanism to differentiate between elements based on their positions in the sequence, which is particularly important in tasks like time series forecasting, where the order of data points carries significant predictive power.

4.2.2 Channel Dependent and Channel Independent Losses

In this section, we adopt the notation from [114] to provide a mathematical analysis of training strategies for multivariate time series forecasting (MTSF). Data in MTSF typically contain multiple variables, or channels, at each time step. The following notations are used in our analysis:

- $X \in \mathbb{R}^{L \times C}$ - the matrix representing historical values of the multivariate time series, where L is the length of the look-back window, and C is the number of channels.
- $Y \in \mathbb{R}^{H \times C}$ - the matrix representing future values to be predicted for the multivariate time series, where H is the forecast horizon.
- ℓ - the loss function, typically the mean squared error (MSE) for regression tasks.

With these notations, we define the losses for the two training strategies:

- **Channel Dependent (CD) loss**

$$\min_f \frac{1}{N} \sum_{i=1}^{N} \ell(f(X^{(i)}), Y^{(i)}) \tag{4.6}$$

- **Channel Independent (CI) loss**

$$\min_f \frac{1}{NC} \sum_{i=1}^{N} \sum_{c=1}^{C} \ell(f(x_c^{(i)}), y_c^{(i)}) \tag{4.7}$$

where $X = [x_1, \cdots, x_C],\ Y = [y_1, \cdots, y_C]$ with $x_c \in \mathbb{R}^L, y_c \in \mathbb{R}^H, 1 \leq c \leq C$.

4.3 Mathematical Analysis

Let's consider a simple linear embedding function for the standard Transformer architecture, which maps the input sequence $X \in \mathbb{R}^{L \times C}$ to a higher-dimensional space:

$$X^E = XW^E \in \mathbb{R}^{L \times e},$$

where e represents the size of the embedded token and $W^E \in \mathbb{R}^{C \times e}$ is a able-to-learn embedding matrix.

With this embedding, the attention mechanism can be calculated as follows:

$$\begin{aligned}\text{Attention}(Q, K, V) &= \text{softmax}\left(\frac{(X^E W^Q)(W^K)^T(X^E)^T}{\sqrt{d_k}}\right) X^E W^V \\ &= \text{softmax}\left(\frac{X(W^E W^Q)(W^K)^T(W^E)^T X^T}{\sqrt{d_k}}\right) XW^E W^V\end{aligned}$$

To simplify the representation, we can replace the matrix multiplications with single matrices, and get:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{XW_1 X^T}{\sqrt{d_k}}\right) XW_2,$$

where $W_1 \in \mathbb{R}^{C \times C}$ and $W_2 \in \mathbb{R}^{C \times O}$ are learnable matrices, and O represents the dimension of output of the attention block.

For the iTransformer [112], the embedding function is given by:

$$X^{iE} = X^T W^{iE} \in \mathbb{R}^{C \times e},$$

where $W^{iE} \in \mathbb{R}^{L \times e}$ is a learnable embedding matrix. Unlike other Transformer-based architectures that embed all dimensions of the input at the same time stamp, the iTrans-

former embeds the entire path for each channel of the input separately. This approach ensures that the embedded vector contains information about the entire sequence for a single channel, allowing the model to capture intricate patterns within each individual channel.

Following a similar approach, we can formulate the attention block in the iTransformer as:

$$\text{iAttention}(Q, K, V) = \text{softmax}\left(\frac{X^T W_3 X}{\sqrt{d_k}}\right) X^T W_4,$$

where $W_3 \in \mathbb{R}^{L\times L}$ and $W_4 \in \mathbb{R}^{L\times O}$ are learnable matrices, and the attention is computed in the channel dimension instead of in the sequence length dimension.

Now, suppose that the test data set has a distribution shift in the first s channels, that is, $X_{\text{test}} = [X_S + \Delta X_S, X_{C-S}] = [x_1 + \Delta x_1, \cdots, x_s + \Delta x_s, x_{s+1}, \cdots, x_C]$. To analyze the impact of this distribution shift on the attention mechanism, we can represent the attention matrix W_1 as four block matrices:

$$W_1 = \begin{pmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{pmatrix} \in \begin{pmatrix} \mathbb{R}^{S\times S} & \mathbb{R}^{S\times(C-S)} \\ \mathbb{R}^{(C-S)\times S} & \mathbb{R}^{(C-S)\times(C-S)} \end{pmatrix}$$

Then, for the standard Transformer attention mechanism, we have:

$$\begin{aligned} \text{Attention}(Q, K, V) =& \text{softmax}\Bigg(\frac{1}{\sqrt{d_k}}\Big((X_S + \Delta X_S)W_{11}(X_S + \Delta X_S)^T \\ &+ X_{C-S}W_{21}(X_S + \Delta X_S)^T + (X_S + \Delta X_S)W_{12}X_{C-S}^T \\ &+ X_{C-S}W_{22}X_{C-S}^T\Big)\Bigg)X_{\text{test}}W_2, \end{aligned}$$

While for the iTransformer attention mechanism, we have:

$$\text{iAttention}(Q,K,V) = \text{softmax}\left(\frac{1}{\sqrt{d_k}}W_{\text{new}}\right)\begin{pmatrix}(X_S+\Delta X_S)^T\\ X_{C-S}^T\end{pmatrix}W_4,$$

$$\text{where}\quad W_{\text{new}} = \begin{pmatrix}(X_S+\Delta X_S)^T W_3(X_S+\Delta X_S) & (X_S+\Delta X_S)^T W_3 X_{C-S}\\ X_{C-S}^T W_3(X_S+\Delta X_S) & X_{C-S}^T W_3 X_{C-S}\end{pmatrix}.$$

From the equations above, we can observe that in the standard Transformer, all attention scores will be affected by the distribution change in the first s channels, and three of the four terms in the attention matrix calculation will change, making the performance unpredictable.

However, for the iTransformer, we can see that for the latter $C-S$ channels that do not have a distribution shift, only the first S elements in the attention matrix calculation will be changed by $(X_{C-S}^T W_3 \cdot \Delta X_S)$. Thus, the attention scores for predicting these channels will be relatively robust to the distribution shift.

This robustness to distribution shifts in a subset of channels gives an explanation for why the iTransformer can outperform existing Transformer-based architectures for time series prediction, especially in tasks where there is an obvious distribution shift from the training dataset to the test dataset.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

This dissertation has presented a comprehensive study and novel contributions in the domains of signature transformations, neural differential equations, and training strategies for multivariate time series forecasting. The key findings and contributions of each chapter are summarized as follows.

- **Chapter 2: Applications of Signature Transformation** - This chapter demonstrated the versatility and efficacy of signature transformations in capturing the essential dynamics of sequential data. The application of signature methods to classification, distribution regression, and other tasks showcased their robustness and flexibility in handling complex time series data.

- **Chapter 3: Applications of Neural Differential Equations** - By exploring the practical applications of Neural Ordinary Differential Equations (NODEs) and Neural Stochastic Differential Equations (NSDEs), this chapter highlighted their power in modeling physical systems and other real-world phenomena. The experi-

ments with high-order NODEs, Neural Manifold ODEs, and neural SDEs, combined with Graph Neural Networks (GNNs) for simulation, demonstrated the significant potential of continuous-depth models in capturing intricate system dynamics.

- **Chapter 4: Training Strategy Analysis—Channel Independence versus Channel Dependence** - This chapter investigated the efficacy of Channel Independent (CI) and Channel Dependent (CD) training strategies in multivariate time series forecasting. The mathematical formulation and theoretical analysis provided insights into why the CI strategy often surpasses the CD strategy. It was revealed that the CI approach's robustness to distribution shifts between training and test datasets is a crucial factor contributing to its superior performance, a common scenario in real-world applications.

Overall, this thesis has provided significant insight into advanced techniques for time series analysis and forecasting, offering theoretical and practical contributions to the field. The comparative analysis of CI and CD strategies, in particular, sheds light on the importance of considering distribution shifts in real-world applications, guiding future research and application development.

5.2 Future Work

While this dissertation has made substantial progress, several avenues for future research remain open:

- **Application of Signature Methods in Reinforcement Learning and Decision-Making Tasks** - Future research could explore the integration of signature methods in reinforcement learning or other decision-making tasks. Given that

signature transformations provide a compact and informative summary of historical data, they could be highly beneficial in environments where a precise model is unnecessary, yet a useful summary of the past is crucial for decision-making.

- **Signature Methods and Channel Categorization** - Address the challenge of long signature vectors in high-dimensional input paths by categorizing channels first. Calculate signatures for subgroups where channels within each group share similar distributions, patterns, or high correlations. This approach reduces the length of the signature vector while leveraging the advantages of the CI strategy.

- **Advancements in Neural Differential Equations** - While this work explored high-order NODEs, Neural Manifold ODEs, and neural SDEs combined with GNNs, future research could further investigate these models' potential. Areas of interest might include exploring higher-dimensional manifolds, incorporating additional physical constraints, or developing more efficient training algorithms. Additionally, applying these models to a broader range of real-world phenomena could further validate their efficacy and reveal new applications. Specifically, adopting a Channel Independent (CI) strategy for NDEs, similar to the channel-wise embedding used in iTransformer, could be explored to improve the training efficiency and model performance.

- **Mathematical Proofs for CI/CD Strategy Comparison** - The comparison between CI and CD training strategies provided valuable insights, but further mathematical proofs are necessary to understand and quantify how distribution shifts affect these strategies. Future research could focus on developing rigorous theoretical frameworks and proofs to better explain the observed phenomena and potentially uncover new training strategies that leverage the strengths of both CI and CD approaches.

- **Hybrid Models and Multi-modal Time Series Analysis** - Combining the strengths of signature transformations, neural differential equations, and transformer-based architectures could lead to hybrid models that leverage the advantages of each method. Future research could explore the design and implementation of such hybrid models, aiming to improve the robustness and accuracy of time series forecasting and other sequential data tasks. Additionally, these hybrid models could be extended to handle multi-modal data sources (e.g., sensor data, text, images, video), developing techniques for fusing and modeling inter-dependencies between different modalities.

- **Scalability and Efficiency** - Investigating ways to enhance the scalability and computational efficiency of the discussed methods is another promising direction. Techniques such as model compression, parallelization, and optimization algorithms could be explored to make these advanced models more practical for large-scale and real-time applications.

- **Improving Interpretability and Trustworthiness** - While deep learning models provide improved predictive accuracy, their "black-box" nature poses challenges for interpretability and trust. Future work could focus on developing methods to make these models more interpretable without compromising performance. This is especially important for critical applications where explainability is crucial.

- **Uncertainty Quantification and Probabilistic Forecasting** - Extend the developed methods to provide uncertainty estimates and probabilistic forecasts, crucial in many real-world applications. Investigate techniques like Bayesian neural networks, ensemble methods, or incorporating domain knowledge to quantify and communicate the uncertainty associated with predictions.

- **Transfer Learning and Domain Adaptation** - Explore transfer learning techniques to leverage knowledge from related domains or tasks, potentially accelerating training and improving performance. Investigate domain adaptation strategies to adapt models to different but related domains, addressing distribution shifts between training and deployment environments. Develop techniques to identify and leverage invariant features or representations across domains.

- **Federated Learning and Privacy-Preserving Techniques** - Investigate federated learning frameworks for training models using decentralized data sources while preserving data privacy. Develop privacy-preserving techniques like differential privacy or secure multi-party computation. Explore trade-offs between privacy, model performance, and computational efficiency.

- **Hardware Acceleration and Edge Computing** - Explore hardware acceleration techniques (e.g., GPUs, TPUs, FPGAs) to improve computational efficiency. Investigate deploying these models on edge devices or resource-constrained environments, leveraging model compression or specialized hardware architectures. Develop edge computing frameworks for real-time decision-making and forecasting in IoT, autonomous systems, and edge analytics.

In conclusion, this dissertation has laid the groundwork for several innovative approaches in time series analysis and forecasting. The insights gained and the methodologies developed provide a solid foundation for future research, with numerous opportunities to further advance the field and address real-world challenges.

Appendix A

Proofs

A.1 HOPE

A.1.1 Proof of Lemma 1

Proof: Assuming $\mathbf{R}^k := (\mathbf{Z}^{t+1} - \mathbf{Z}^t)/\sqrt{\lambda}$ and $\beta := 1 - \gamma\sqrt{\lambda}$ where γ is a given coefficient, the momentum updating algorithm can be rewritten as follows:

$$\mathbf{R}^{t+1} = (1 - \gamma\sqrt{\lambda})\mathbf{R}^t + \sqrt{\lambda}(\sigma((\hat{\mathbf{D}}^t)^{-1}\hat{\mathbf{A}}^t\mathbf{Z}^t\mathbf{W}_p) - \mathbf{Z}^t). \tag{A.1}$$

When $\lambda \to 0$, we have

$$\frac{d\mathbf{Z}^t}{dt} = \mathbf{R}^t, \tag{A.2}$$

$$\frac{d\mathbf{R}^t}{dt} = -\gamma\mathbf{R}^t + (\sigma((\hat{\mathbf{D}}^t)^{-1}\hat{\mathbf{A}}^t\mathbf{Z}^t\mathbf{W}_p) - \mathbf{Z}^t), \tag{A.3}$$

Finally, we combine Equation A.2 and Equation A.3 by deleting $\mathbf{R}^t$ as below:

$$\frac{d^2\mathbf{Z}^t}{dt^2} + \gamma\frac{d\mathbf{Z}^t}{dt} = \sigma((\hat{\mathbf{D}}^t)^{-1}\hat{\mathbf{A}}^t\mathbf{Z}^t\mathbf{W}_p) - \mathbf{Z}^t. \tag{A.4}$$

A.1.2 Proof of Lemma 2

Proof: A coupled first-order ODE can be directly obtained from Equation A.2 and Equation A.3. Further, we can augment the node state representation matrix by $\mathbf{F}^t = [\mathbf{Z}^t, \dot{\mathbf{Z}}^t]$ and $\dot{\mathbf{F}}^t = [\dot{\mathbf{Z}}^t, \ddot{\mathbf{Z}}^t]$[1], resulting in a first-order ODE with the variable $\mathbf{F}^t$.

A.1.3 Proof of Lemma 3

Proof: Let the message passing matrix be

$$\mathbf{M}^t = (\hat{\mathbf{D}}^t)^{-1}\hat{\mathbf{A}}^t = \begin{pmatrix} M_{11}^t & \dots & M_{1N}^t \\ \vdots & \ddots & \vdots \\ M_{N1}^t & \dots & M_{NN}^t \end{pmatrix}, \tag{A.5}$$

the weight matrix is

$$\mathbf{W}_p = \begin{pmatrix} W_{11} & \dots & W_{1d} \\ \vdots & \ddots & \vdots \\ W_{d1} & \dots & W_{dd} \end{pmatrix}. \tag{A.6}$$

Then, we define the transformation function

$$S\colon \mathbb{R}^{N\times d} \to \mathbb{R}^{Nd\times 1} \tag{A.7}$$

$$\mathbf{Z}^t = \begin{pmatrix} \mathbf{z}_1^t \\ \vdots \\ \mathbf{z}_N^t \end{pmatrix} \mapsto \begin{pmatrix} (\mathbf{z}_1^t)^T \\ \vdots \\ (\mathbf{z}_N^t)^T \end{pmatrix}, \tag{A.8}$$

[1] $\dot{\mathbf{Z}}^t := \frac{d\mathbf{Z}^t}{dt}$ and $\ddot{\mathbf{Z}}^t := \frac{d^2\mathbf{Z}^t}{dt^2}$

and transform the augmented ODE Equations A.2 and A.3 to be

$$\begin{cases} \frac{d}{dt}S(\mathbf{Z}^t) = S(\mathbf{R}^t), \\ \frac{d}{dt}S(\mathbf{R}^t) = -\gamma S(\mathbf{R}^t) - S(\mathbf{Z}^t) + \mathbf{ReLU}(S(\mathbf{M}^t\mathbf{R}^t\mathbf{W}_p)) \end{cases} \tag{A.9}$$

where $(\mathbf{M}^t\mathbf{R}^t\mathbf{W}_p)_{ij} = \sum_{b=1}^{d}\sum_{a=1}^{N} M_{ia}^t Z_{ab}^t W_{bj}$.

Now, let $\mathbf{Y}^t = \begin{pmatrix} S(\mathbf{Z}^t) \\ S(\mathbf{R}^t) \end{pmatrix} \in \mathbb{R}^{2Nd\times 1}$, we get the ODE equation $\frac{d\mathbf{Y}^t}{dt} = f(\mathbf{Y}^t)$. It is obvious that f is continuous w.r.t t since it does not depend on t explicitly.

For any two solutions $\mathbf{Y}_1^t, \mathbf{Y}_2^t$, denote $\Delta R_i^t = S(\mathbf{R}^t)_{1i} - S(\mathbf{R}^t)_{2i}$, $\Delta Z_i^t = S(\mathbf{Z}^t)_{1i} - S(\mathbf{Z}^t)_{2i}$. Note $|\mathbf{ReLU}(x) - \mathbf{ReLU}(y)| = |x^+ - y^+| \le |x-y|$. Therefore, we have

$$\begin{aligned} \|f(\mathbf{Y}_1^t) - f(\mathbf{Y}_2^t)\|_2^2 &\le \sum_{i=1}^{Nd}(\Delta R_i^t)^2 && \text{(A.10)} \\ &\quad + 3\sum_{i=1}^{Nd}\big[\gamma^2(\Delta R_i^t)^2 + (\Delta S_i^t)^2 && \text{(A.11)} \\ &\quad + N^2d^2\sum_{a,b}(M_{ka}^t)^2 W_{bj}^2(\mathbf{Z}_{ab,1}^t - \mathbf{Z}_{ab,2}^t)^2\big] && \text{(A.12)} \\ &\le \sum_{i=1}^{Nd}\big[(1+3\gamma^2)(\Delta R_i^t)^2 && \text{(A.13)} \\ &\quad + (3+3N^3d^3M^2W^2)(\Delta Z_i^t)^2\big] && \text{(A.14)} \\ &\le L^2\|\mathbf{Y}_1^t - \mathbf{Y}_2^t\|_2^2 && \text{(A.15)} \end{aligned}$$

where $M = \max_{i,j}|M_{ij}^t|$, $W = \max_{ij}|W_{ij}|$, $L = \max(\sqrt{1+3\gamma^2}, \sqrt{3+3N^3d^3M^2W^2})$. Hence, we prove that f is Lipschitz-continuous in y, then by Picard–Lindelöf theorem 5, we prove the uniqueness of the solution.

A.2 CARE

A.2.1 Proof of Lemma 4

Proof: We have:

$$
\begin{aligned}
&\mathrm{P}\left(\mathbf{Y}^t \mid G^{0:t-1}\right) \\
&= \int \mathrm{P}\left(\mathbf{Y}^t \mid \mathbf{c}^{t-1}, G^{0:t-1}\right) \cdot \mathrm{P}\left(\mathbf{c}^{t-1} \mid G^{0:t-1}\right) d\mathbf{c}^{t-1} \\
&= \int \mathrm{P}\left(\mathbf{Y}^t \mid \mathbf{c}^{t-1}, G^{t-1}\right) \cdot \mathrm{P}\left(\mathbf{c}^{t-1} \mid G^{0:t-1}\right) d\mathbf{c}^{t-1} \\
&= \int \mathrm{P}\left(\mathbf{Y}^t \mid \mathbf{c}^{t-1}, G^{t-1}\right) \cdot \mathrm{P}\left(\mathbf{c}^{t-1} \mid \mathbf{c}^{t-k}, G^{0:t-1}\right) \cdot \mathrm{P}\left(\mathbf{c}^{t-k} \mid G^{0:t-k}\right) d\mathbf{c}^{t-1} d\mathbf{c}^{t-k} \\
&= \int \mathrm{P}\left(\mathbf{Y}^t \mid \mathbf{c}^{t-1}, G^{t-1}\right) \cdot \mathrm{P}\left(\mathbf{c}^{t-1} \mid \mathbf{c}^{t-k}, G^{t-k:t-1}\right) \cdot \mathrm{P}\left(\mathbf{c}^{t-k} \mid G^{0:t-k}\right) d\mathbf{c}^{t-1} d\mathbf{c}^{t-k}
\end{aligned}
\tag{A.16}
$$

A.2.2 Proof of Lemma 5

For convenience, Eqn. 3.49 in the main paper is repeated as:

$$
\frac{d\mathbf{v}_i^s}{ds} = \Phi([\mathbf{v}_1^s, \cdots, \mathbf{v}_N^s, \mathbf{c}^s]) = \sigma\left(\sum_{j \in \mathcal{N}^s(i)} \frac{\hat{\mathbf{A}}_{ij}^s}{\sqrt{\hat{D}_i^s \cdot \hat{D}_j^s}} \mathbf{v}_j^s \mathbf{W}_1 + \mathbf{c}^s \mathbf{W}_2\right), \tag{A.17}
$$

Eqn. 3.51 is repeated as:

$$
\frac{d\mathbf{c}^s}{ds} = \sigma\left(\sum_{i=1}^{N} (\mathbf{v}_i^s \mathbf{W}_3 + \frac{d\mathbf{v}_i^s}{ds} \mathbf{W}_4) + \mathbf{c}^s \mathbf{W}_5\right). \tag{A.18}
$$

Proof: Let $\mathbf{A}_{ij}^s = 0$ if $j \notin \mathcal{N}^s(i)$ and denote $\mathbf{M}_{ij}^s = \frac{\hat{\mathbf{A}}_{ij}^s}{\sqrt{\hat{D}_i^s \cdot \hat{D}_j^s}}$. Then, we transpose

them with Eqn. A.17 and A.18 becoming:

$$\begin{aligned}\frac{d(\mathbf{v}_i^s)^T}{ds} &= \sigma\left(\sum_{j=1}^{N}\mathbf{M}_{ij}^s\mathbf{W}_1^T(\mathbf{v}_j^s)^T + \mathbf{W}_2^T(\mathbf{c}^s)^T\right)\\ \frac{d(\mathbf{c}^s)^T}{ds} &= \sigma\left(\sum_{i=1}^{N}(\mathbf{W}_3^T(\mathbf{v}_i^s)^T + \mathbf{W}_4^T\frac{d(\mathbf{v}_i^s)^T}{ds}) + \mathbf{W}_5^T(\mathbf{c}^s)^T\right)\\ &= \sigma\left(\sum_{j=1}^{N}(\mathbf{W}_3^T + (\sum_{i=1}^{N}\mathbf{M}_{ij}^s)\mathbf{W}_4^T\mathbf{W}_1^T)(\mathbf{v}_j^s)^T + (\mathbf{W}_5^T + N\mathbf{W}_4^T\mathbf{W}_2^T)(\mathbf{c}^s)^T\right).\end{aligned} \tag{A.19}$$

Denote $\mathbf{S}_j^s = \mathbf{W}_3^T + (\sum_{i=1}^{N}\mathbf{M}_{ij}^s)\mathbf{W}_4^T\mathbf{W}_1^T$ and $\mathbf{S}_c = \mathbf{W}_5^T + N\mathbf{W}_4^T\mathbf{W}_2^T$. Then, we can get the following ODE system:

$$\frac{d}{ds}\begin{pmatrix}(\mathbf{v}_1^s)^T\\ \vdots\\ (\mathbf{v}_N^s)^T\\ (\mathbf{c}^s)^T\end{pmatrix} = \begin{pmatrix}\mathbf{M}_{11}^s\mathbf{W}_1^T & \cdots & \mathbf{M}_{1N}^s\mathbf{W}_1^T & \mathbf{W}_2^T\\ \vdots & \ddots & \vdots & \vdots\\ \mathbf{M}_{N1}^s\mathbf{W}_1^T & \cdots & \mathbf{M}_{NN}^s\mathbf{W}_1^T & \mathbf{W}_2^T\\ \mathbf{S}_1^s & \cdots & \mathbf{S}_N^s & \mathbf{S}_c\end{pmatrix}\begin{pmatrix}(\mathbf{v}_1^s)^T\\ \vdots\\ (\mathbf{v}_N^s)^T\\ (\mathbf{c}^s)^T\end{pmatrix}. \tag{A.20}$$

Now, let $\mathbf{Y}^s = \begin{pmatrix}(\mathbf{v}_1^s)^T\\ \vdots\\ (\mathbf{v}_N^s)^T\\ (\mathbf{c}^s)^T\end{pmatrix} \in \mathbf{R}^{(N\times d_v+d_c)\times 1}$, where $\mathbf{v}_i \in \mathbf{R}^{d_v}, \mathbf{c} \in \mathbf{R}^{d_c}$. For the ODE equation $\frac{d\mathbf{Y}^s}{ds} = \mathbf{F}^s\mathbf{Y}^s$, with Assumption 8, the coefficient matrix $\mathbf{F}^t$ is continuous w.r.t t since all coefficients in the matrix are continuous w.r.t t.

For any two solutions $\mathbf{Y}_1^t, \mathbf{Y}_2^t$, we have:

$$\|f(\mathbf{Y}_1^t) - f(\mathbf{Y}_2^t)\|_2 \le L\|\mathbf{Y}_1^t - \mathbf{Y}_2^t\|_2, \tag{A.21}$$

where $L = \max_{i,j}|\mathbf{F}_{ij}^t| \le W + NCW^2$. Then by Picard–Lindelöf theorem 5, we prove

the uniqueness of the solution.

A.3 POEM

A.3.1 Proof of Lemma 6

For convenience, Eqn. 3.57 in the main paper is repeated as:

$$\frac{d\mathbf{h}_i^{s,E}}{ds} = \log_{\mathbf{h}_i^t}(f_{\mathbf{h}_i^t}(\exp_{\mathbf{h}_i^t}(\mathcal{N}^l(\mathbf{h}_i^s) + \mathcal{N}^g(\mathbf{h}_i^s)))), \tag{A.22}$$

Proof: By the definition of $f_{\mathbf{h}_i^s}$, the ODE system can by simplified to $\frac{d\mathbf{h}_i^{s,E}}{ds} = \log_{\mathbf{h}_i^t}(\exp_{\mathbf{h}_i^s} f_{\text{pro}}(\mathcal{N}(\mathbf{h}_i^s)))$ and Eqn. A.22 becomes $\frac{d\mathbf{h}_i^{s,E}}{ds} = \log_{\mathbf{h}_i^t}(\exp_{\mathbf{h}_i^t} f_{\text{pro}}(\mathcal{N}(\mathbf{h}_i^s)))$. Denote the solutions by $\mathbf{H}_{i,1}^s$ and $\mathbf{H}_{i,2}^s$ respectively.

Both exponential and logarithmic maps are smooth, i.e, $\forall\varepsilon > 0, \exists\delta_1(\varepsilon), \delta_2(\varepsilon)$, when $\|\mathbf{x}_1 - \mathbf{x}_2\|_{L^2} \leq \delta_1, \|\mathbf{y}_1 - \mathbf{y}_2\|_{L^2} \leq \delta_2$, we have

$$\|\exp_{\mathbf{x}_1}(\mathbf{h}) - \exp_{\mathbf{x}_2}(\mathbf{h})\|_{L^2} \leq \varepsilon, \|\log_{\mathbf{x}}(\mathbf{y}_1) - \log_{\mathbf{x}}(\mathbf{y}_2)\|_{L^2} \leq \varepsilon. \tag{A.23}$$

Therefore, we only need to choose $\Delta t < \frac{\delta_1(\delta_2)}{C_1}$, with the assumption of Lipschitz-continuity, we can get $\|\exp_{\mathbf{h}_i^s} f_{\text{pro}}(\mathcal{N}(\mathbf{h}_i^s)) - \exp_{\mathbf{h}_i^t} f_{\text{pro}}(\mathcal{N}(\mathbf{h}_i^s))\|_{L^2} \leq \delta_2$, thus

$$\|\log_{\mathbf{h}_i^t}(\exp_{\mathbf{h}_i^s} f_{\text{pro}}(\mathcal{N}(\mathbf{h}_i^s))) - \log_{\mathbf{h}_i^t}(\exp_{\mathbf{h}_i^t} f_{\text{pro}}(\mathcal{N}(\mathbf{h}_i^s)))\|_{L^2} \leq \varepsilon \tag{A.24}$$

It means for any $\varepsilon > 0, s \in [t, t + \Delta t]$, the difference between solutions satisfies $\left\|\frac{d(\mathbf{H}_{i,1}^s - \mathbf{H}_{i,2}^s)}{ds}\right\|_{L^2} \leq \varepsilon$. As a result,

$$\|\mathbf{H}_{i,1}^s - \mathbf{H}_{i,2}^s\|_{L^2} \leq \varepsilon \cdot \Delta t$$

A.3.2 Proof of Corollary 13

For convenience, Eqn. 3.54 in the main paper is repeated as:

$$\frac{d\mathbf{h}_i^s}{ds} = \log_{\mathbf{h}_i^s}(f_{\mathbf{h}_i^s}(\mathcal{N}(\mathbf{h}_i^s))) \in \mathcal{T}_{\mathbf{h}_i^s}\mathcal{M}, \forall i. \tag{A.25}$$

Proof: Suppose we have $\mathbf{h} = (\mathbf{h}_0, \mathbf{h}_1, \cdots, \mathbf{h}_n)$, then the directional derivative of the exponential map can be expressed as

$$\frac{\partial \exp_{\mathbf{x}}(\mathbf{h})}{\partial \mathbf{h}} = \Big(\frac{\partial \exp_{\mathbf{x}}(\mathbf{h})}{\partial \mathbf{h}_0}, \frac{\partial \exp_{\mathbf{x}}(\mathbf{h})}{\partial \mathbf{h}_1}, \cdots, \frac{\partial \exp_{\mathbf{x}}(\mathbf{h})}{\partial \mathbf{h}_n}\Big) \tag{A.26}$$

$$\frac{\partial \exp_{\mathbf{x}}(\mathbf{h})}{\partial \mathbf{h}_0} = \cosh(\|\mathbf{v}\|_{\mathcal{L}})\,\mathbf{h}_0 + \sinh(\|\mathbf{v}\|_{\mathcal{L}})\,\mathbf{x} \tag{A.27}$$

$$\frac{\partial \exp_{\mathbf{x}}(\mathbf{h})}{\partial \mathbf{h}_j} = \frac{\sinh(\|\mathbf{v}\|_{\mathcal{L}})}{\|\mathbf{v}\|_{\mathcal{L}}}\mathbf{h}_j,\ \forall j \geq 1 \tag{A.28}$$

Now, suppose we have solutions to Eqn. A.25 and $\frac{d\mathbf{h}_i^{s,E}}{ds} = \log_{\mathbf{h}_i^t} f_{\mathbf{h}_i^s}(\mathcal{N}(\mathbf{h}_i^s))$ denoted by $\mathbf{H}_{i,1}^s$ and $\mathbf{H}_{i,2}^s$ respectively with the same initial state $\mathbf{H}_i^t$. Then by Cauchy-Schwarz inequality, we have

$$\Big\|\frac{d(\mathbf{H}_{i,1}^s - \mathbf{H}_{i,2}^s)}{ds}\Big\|_{L^2} = \Big\|\log_{\mathbf{h}_i^s} f_{\mathbf{h}_i^s}(\mathcal{N}(\mathbf{h}_i^s)) - \frac{\partial \exp_{\mathbf{H}_i^t}(\mathbf{H}_{i,2}^{s,E})}{\partial \mathbf{H}_{i,2}^{s,E}} \log_{\mathbf{h}_i^t} f_{\mathbf{h}_i^s}(\mathcal{N}(\mathbf{h}_i^s))\Big\|_{L^2} \tag{A.29}$$

$$\leq \Big\|I_{(n+1)\times(n+1)} - \frac{\partial \exp_{\mathbf{H}_i^t}(\mathbf{H}_{i,2}^{s,E})}{\partial \mathbf{H}_{i,2}^{s,E}}\Big\|_1 \cdot \tag{A.30}$$

$$\Big\|\log_{\mathbf{h}_i^s} f_{\mathbf{h}_i^s}(\mathcal{N}(\mathbf{h}_i^s)) - \log_{\mathbf{h}_i^t} f_{\mathbf{h}_i^s}(\mathcal{N}(\mathbf{h}_i^s))\Big\|_{L^2} \tag{A.31}$$

By the assumption that mesh representations are bounded by the Lorentz norm, the matrix norm will be bounded by some constant C. By the continuity of the logarithmic map, the vector norm is bounded by ε. Therefore, we get $\big\|\frac{d(\mathbf{H}_{i,1}^s - \mathbf{H}_{i,2}^s)}{ds}\big\|_{L^2} \leq C\varepsilon$.

Denote the solution to Eqn. A.22 by $\mathbf{H}_{i,3}^s$ with the initial state $\mathbf{H}_i^t$. Then by Lemma

6 we have

$$\left\|\frac{d(\mathbf{H}^s_{i,1}-\mathbf{H}^s_{i,3})}{ds}\right\|_{L^2} \leq \left\|\frac{d(\mathbf{H}^s_{i,1}-\mathbf{H}^s_{i,2})}{ds}\right\|_{L^2} + \left\|\frac{d(\mathbf{H}^s_{i,2}-\mathbf{H}^s_{i,3})}{ds}\right\|_{L^2} \tag{A.32}$$

$$\leq (C+1)\varepsilon. \tag{A.33}$$

As a result, we have the difference between solutions to Eqn. A.25 and Eqn. A.22 is bounded by $(C+1)\varepsilon \cdot \Delta t$ in the interval $[t, t+\Delta t]$.

A.4 GraphSDE

A.4.1 Proof of Lemma 7

Proof: Define $V(\varepsilon, t) = \|\varepsilon\|^2$, a non-negative real valued function defined on $\mathbb{R}^{3dN} \times \mathbb{R}^+$. Recall $\varepsilon^t = \mathbf{h}^t_p - \mathbf{h}^t$, where $\mathbf{h}^t_p$ is the perturbed solution; and $d\varepsilon^t = f_\Delta(\varepsilon^t, t)dt + g_\Delta(\varepsilon^t, t)d\mathbf{W}^t$, where

$$f_\Delta(\varepsilon^t, t) = \phi^{dr}_c\left(\mathbf{h}^t_p, t\right) - \phi^{dr}_c\left(\mathbf{h}^t, t\right) = \mathbf{M}^{t,dr}\varepsilon^t;$$

$$g_\Delta(\varepsilon^t, t) = \phi^{di}_c\left(\mathbf{h}^t_p, t\right) - \phi^{di}_c\left(\mathbf{h}^t, t\right) = \mathbf{M}^{t,di}\varepsilon^t.$$

Here we assume the Brownian motions for the SDE associated with $\mathbf{h}^0_p$ and $\mathbf{h}^0$ has the same sample path. By Eqn. 3.67, it can be obtained that f_Δ and g_Δ are at most liner:

$$\begin{aligned} \|f_\Delta(\varepsilon^t, t)\| &= \|\mathbf{M}^{dr}_t\varepsilon^t\| \leq \lambda_{\max}\|\varepsilon^t\| < \lambda_{\max}(1+\|\varepsilon^t\|) \\ \|g_\Delta(\varepsilon^t, t)\| &= \|\mathbf{M}^{di}_t\varepsilon^t\| \leq \lambda_{\max}\|\varepsilon^t\| < \lambda_{\max}(1+\|\varepsilon^t\|); \end{aligned} \tag{A.34}$$

and Lipschitz continuous:

$$\begin{aligned}\|f_\Delta(\varepsilon_1^t,t)-f_\Delta(\varepsilon_2^t,t)\| &= \|\mathbf{M}_t^{dr}(\varepsilon_1^t-\varepsilon_2^t)\| \le \lambda_{\max}\|\varepsilon_1^t-\varepsilon_2^t\|;\\ \|g_\Delta(\varepsilon_1^t,t)-g_\Delta(\varepsilon_2^t,t)\| &= \|\mathbf{M}_t^{di}(\varepsilon_1^t-\varepsilon_2^t)\| \le \lambda_{\max}\|\varepsilon_1^t-\varepsilon_2^t\|.\end{aligned} \tag{A.35}$$

By Lemma 3.3 in [100], $\mathbb{P}\{\varepsilon^t \neq \mathbf{0} \text{ for all } t \ge 0\}$ for all $\varepsilon^0 \neq \mathbf{0}$. The second order Taylor series of $V(\varepsilon,t)$ is bounded by

$$\begin{aligned}&\frac{\partial V(\varepsilon,t)}{\partial \varepsilon} f_\Delta(\varepsilon,t) + \frac{1}{2}\operatorname{Tr}\left[g_\Delta^\top(\varepsilon_1^t,t)\frac{\partial^2 V(\varepsilon,t)}{\partial\varepsilon\partial\varepsilon^\top} g_\Delta(\varepsilon_1^t,t)\right]\\ \le& 2\|\varepsilon\|\cdot\lambda_{\max}\|\varepsilon\| + \lambda_{\max}^2\|\varepsilon\|^2 = (\lambda_{\max}^2+2\lambda_{\max})\|\varepsilon\|^2.\end{aligned} \tag{A.36}$$

Note that Eqn. A.36 remains valid for $\varepsilon = \mathbf{0}$. Applying Itô's formula with Eqn. A.36, for all $t \ge 0$,

$$\begin{aligned}\log V\left(\varepsilon^t,t\right) \le& \log V\left(\varepsilon^0,0\right) + (\lambda_{\max}^2+2\lambda_{\max})\|\varepsilon\|^2 t + M(t)\\ &- \frac{1}{2}\int_0^t \frac{\left|\frac{\partial V(\varepsilon,t)}{\partial\varepsilon}\Big|_{\varepsilon=\varepsilon^s}\boldsymbol{G}_\Delta\left(\varepsilon^s,s\right)\right|^2}{V^2\left(\varepsilon^s,s\right)}ds,\end{aligned} \tag{A.37}$$

where $M(t) = \int_0^t \frac{\left|\frac{\partial V(\varepsilon,t)}{\partial\varepsilon}\Big|_{\varepsilon=\varepsilon^s} g_\Delta(\varepsilon^s,s)\right|^2}{V^2(\varepsilon^s,s)} d\mathbf{W}_s$ is a continuous martingale with initial $M(0) = 0$. By the exponential martingale inequality, for any arbitrary $\alpha \in (0,1)$ and $n \in \mathbb{N}_+$,

$$\mathbb{P}\left\{\sup_{0\le t\le n}\left[M(t) - \frac{\alpha}{2}\int_0^t \frac{\left|\frac{\partial V(\varepsilon,t)}{\partial\varepsilon}\Big|_{\varepsilon=\varepsilon^s} g_\Delta\left(\varepsilon^s,s\right)\right|^2}{V^2\left(\varepsilon^s,s\right)}\mathrm{d}s\right] > \frac{2}{\alpha}\log n\right\} \le \frac{1}{n^2}. \tag{A.38}$$

By Borel–Cantelli lemma, there exists an integer n_0 such that $\forall n \ge n_0$,

$$M(t) \le \frac{2}{\alpha}\log n + \frac{\alpha}{2}\int_0^t \frac{\left|\frac{\partial V(\varepsilon,t)}{\partial\varepsilon}\Big|_{\varepsilon=\varepsilon^s} g_\Delta\left(\varepsilon^s,s\right)\right|^2}{V^2\left(\varepsilon^s,s\right)}ds, \quad \forall 0 \le t \le n. \tag{A.39}$$

Combine Eqn. A.36, Eqn. A.39 and

$$\left\| \frac{\partial V(\varepsilon, t)}{\partial \varepsilon}\bigg|_{\varepsilon=\varepsilon^s} g_\Delta\left(\varepsilon^s, s\right) \right\|^2 = 4\|\varepsilon^{s\top}\mathbf{M}_t^{di}\varepsilon^s\|^2 \geq 4\lambda_{\min}^2 V^2(\varepsilon^s, t),$$

one obtain

$$\log V\left(\varepsilon^t, t\right) \leq \log V\left(\varepsilon^0, 0\right) - \frac{1}{2}\left[(1-\alpha)4\lambda_{\min}^2 - 2(\lambda_{\max}^2 + 2\lambda_{\max})\right] t + \frac{2}{\alpha}\log n. \quad \text{(A.40)}$$

Thus, for all $n-1 \leq t \leq n$ and $n \geq n_0$,

$$\frac{1}{t}\log V\left(\varepsilon^t, t\right) \leq -\frac{1}{2}\left[(1-\alpha)4\lambda_{\min}^2 - 2(\lambda_{\max}^2 + 2\lambda_{\max})\right] + \frac{\log V\left(\varepsilon^0, 0\right) + \frac{2}{\alpha}\log n}{n-1}$$

which implies:

$$\limsup_{t\to\infty} \frac{1}{t}\log V\left(\varepsilon_t, t\right) \leq -\frac{1}{2}\left[(1-\alpha)4\lambda_{\min}^2 - 2(\lambda_{\max}^2 + 2\lambda_{\max})\right] \quad \text{a.s.} \quad \text{(A.41)}$$

Therefore we only need to guarantee $2\lambda_{\min}^2 \geq \lambda_{\max}^2 + 2\lambda_{\max}$, we can have almost surely exponential stability. After solve the equation, we get the relation $2 \leq \lambda_{\min} \leq \lambda_{\max} \leq \sqrt{2\lambda_{\min}^2 + 1} - 1$. Of course, it's a rough estimation, there might exist other choices for the message passing matrix to help guarantee the almost surely exponential stability.

A.4.2 Proof of Theorem 14

Proof: We start with the case $t = 0$. For simplification, we define $\Delta_z^0(\mathbf{y}) := g^0(\mathbf{y}) - g^0(\mu(\mathbf{X}^0))$. To compare $\mathcal{R}_{\mathcal{X}}^r(0)$ and $\tilde{\mathcal{R}}_{\mathcal{X}}^r(0)$, we first identify $\mathcal{R}_z^r(0)$ satisfying:

$$\mathbb{P}(\|\Delta_Z^0(\mathbf{Z}^0 + \delta_{\mathbf{Z}^0})\| \leq r) > \mathbb{P}(\|\Delta_Z^0(\mathbf{Z}^0 + \delta_{\mathbf{Z}^0})\| > r). \quad \text{(A.42)}$$

Let $\mathcal{A}^r(0)$ be the set of all $\delta_{\mathbf{Z^0}}$ for which Equation Eqn. A.42 holds. We define

$$p_\Delta(\delta_{\mathbf{Z^0}}) := \mathbb{P}(\|\Delta_z^0(\mathbf{Z}^0 + \delta_{\mathbf{Z}^0})\| \leq r). \tag{A.43}$$

Given $\mathbf{Z}^0 = \mu(\mathbf{X}^0) + \eta \circ \sigma(\mathbf{X}^0)$ is Gaussian conditioned on $\mu(\cdot)$ and $\sigma(\cdot)$, $p_\Delta(\delta_{\mathbf{Z^0}})$ becomes a continuous function of $\delta_{\mathbf{Z^0}}$.

Applying the Neyman-Pearson lemma [115], we obtain that the boundary between $\mathcal{A}^r(0)$ and its complement is a hyperplane perpendicular to the direction of lowest variance in $\mathbf{Z}^0$. In this direction, δ_{Z^0} increases. Following a similar argument as in [116],

$$\|\delta_{Z^0}\| = \left(\min \sigma(\mathbf{X}^0)\right) \Phi^{-1} \left(\mathbb{P}\left(\|\Delta(\mathbf{X}^0)\|_2 \leq r\right)\right), \tag{A.44}$$

where Φ is the cumulative distribution function of the standard normal distribution, and $\Delta(\mathbf{X}^0) = g^0(\mathbf{Z}^0) - g^0(\mu(\mathbf{X}))$.

According to Lemma 7, $\mathbf{Z}^t$ from Equation 3.64 is almost surely exponentially stable, implying

$$\|g^t(\mathbf{Z}^t) - g^t(\mu(\mathbf{X}^t))\| \leq c_2 \|\varepsilon^t\| \to 0. \tag{A.45}$$

Conversely, when $\mathbf{Z}^t$ originates from the model in Equation 3.64 minus the diffusion term $\int \phi^{di}(\mathbf{Z}^s, s) d\mathbf{W}^s$, [100] shows that $\mathbf{Z}^t$ is not stable. In this scenario, $\Delta(\mathbf{X^t}) > c_3 > 0$, making $\Phi^{-1}(\mathbb{P}(\|\Delta(\mathbf{X^0})\|_2 \leq r))$ larger for the SDE model compared to the ODE model. Thus we obtain $\tilde{\mathcal{R}}^r_{\mathcal{X}}(0) < \mathcal{R}^r_{\mathcal{X}}(0)$. The proof is complete by combing Eqn. A.45, the SDE model Eqn. 3.64 and the Lipschitz continuity of g^t.

Appendix B

Experiment Details

B.1 HOPE

B.1.1 Details of Datasets

We utilize the COVID-19 data from the Johns Hopkins University (JHU) Center for Systems Science and Engineering [117] to build our node feature data. We selected five dynamic features along with one static feature as object attributes. Detailed information about these features is introduced below.

- **Population**: The number of population in each state.
- **Confirmed-Number**: The number of state increased confirmed cases in each day.
- **Deaths-Number**: The number of state increased deaths in each day.
- **Recoverd-Number**: The number of state increased recovered cases in each day.
- **Mortality-Rate**: The number of state cumulative deaths / the number of state cumulative confirmed cases each day.

- **Testing-Rate**: The number of state cumulative test results per 100,000 persons in each day.

Then we manage to generate training and testing samples. To capture dynamic spatial correlations between each object, the Dynamic Time Warping (DTW) algorithm [118] is employed for similarity measurement of states. To be specific, for each time t the edge weight of two nodes is measured with their feature series in $[t - \Delta, t]$ by the DTW algorithm. **Social Network** models opinions migrating from individuals to individuals in a social network. Following [119], the number of individuals and the noise parameter is set to 80 and 0.2, respectively. The sparsity parameter is set to $e^{-0.4}$. For **Spring Ocsillator**, we set the number of balls to 50 and simulate the data for total 240 timestamps. The side length of the box is set to 2 and the initial locations of these balls follow uniform distribution in the area inside the box. Every two balls have a probability of 0.5 to be connected together with a spring, and we also ensure that every ball has at least one spring on it. As we discussed before, each training or testing sample is a continuous time series composed of the condition part and prediction part. The conditional part is model input and the prediction part is used for supervising or evaluating. As a result, it is sufficient to ensure no overlapping between the training sample and the testing sample.

To be specific, we split feature data in COVID-19, a 266-days time series, into a 233-day part and 31-day part. The training samples and validating samples are extracted from 233-day part, and testing samples are extracted from 31-day part. The similar procedure is deployed on Social Network and Spring Oscillator. Socail Network is divided into a 320-day part and a 80-day part, and Spring Oscillator is divided into a 500-stamp part and a 50-stamp part.

To evaluate our HOPE and baselines, we select several testing samples and adopt the

average performance among the samples. For example, on the 2-week-ahead prediction task in COVID-19, we select Dec.02-Dec.15, Dec.10-Dec.22, Dec.17-Dec.29 as testing samples. The three metrics, i.e., MAE, RMSE and MAPE are then computed on each sample. Finally, we take the average on these samples to report.

B.1.2 Details of Baselines

The details of baselines are elaborated as follows:

- **LSTM** [49]: It is a classic recurrent neural network (RNN) that learns the dynamics of the sequence, without considering the interaction between nodes.
- **GRU** [53]: It is another variant of RNNs, which involves two gates to model temporal evolution.
- **NODE** [61]: It is the first continuous-depth neural network model which is solved by the back-propagatable ODE solver.
- **HBNODE** [120]: It employs a heavy ball ODE to accelerate the forward and back propagation of the ODE model.
- **DGCRN** [121]: It constructs the dynamic graph and utilizes a graph convolution recurrent unit to capture the real-time spatial-temporal dependencies in the dynamical system.
- **MPNODE** [110]: It combines augmented ODE with message passing mechanism.
- **CG-ODE** [119]: It is a graph ODE model that integrates the evolution of both edges and nodes into a holistic ODE system.

B.2 CARE

B.2.1 Dataset Details

We evaluate our proposed CARE on four physical simulation datasets with temporal distribution shift caused by environmental variations. Then we introduce the details of these four datasets.

- *Lennard-Jones Potential* (a.k.a. 6-12 potential) is popular in modeling electronically neutral atoms or molecules, which can be formulated as:

$$V_{\mathrm{LJ}} = 4\varepsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^{6} \right], \tag{B.1}$$

  where r is the distance between particle pairs, σ denotes the size of the particle, ϵ denotes the depth of the potential well. The first term denotes the attractive force, which decreases as the distance between particles increases. The second term denotes the repulsive force, which increases when two particles are too close. The temperature in the system is changed along with the time to model the environmental variations and a high temperature would bring a more intense molecular motion.

- *3-body Stillinger-Weber Potential* provides more complex relationships besides pairwise relationships in *Lennard-Jones Potential*. It contains both two-body and three-body terms with the following formulation:

$$V_{\mathrm{SW}} = \sum_{i} \sum_{j>i} \phi_2 \left(r_{ij} \right) + \sum_{i} \sum_{j \neq i} \sum_{k>j} \phi_3 \left(r_{ij}, r_{ik}, \theta_{ijk} \right), \tag{B.2}$$

  where $\phi_2 \left(r_{ij} \right) = A_{ij} \epsilon_{ij} \left[B_{ij} \left(\frac{\sigma_{ij}}{r_{ij}} \right)^{p_{ij}} - \left(\frac{\sigma_{ij}}{r_{ij}} \right)^{q_{ij}} \right] \exp \left(\frac{\sigma_{ij}}{r_{ij} - a_{ij} \sigma_{ij}} \right)$ is the two-body term

and $\phi_3\left(r_{ij}, r_{ik}, \theta_{ijk}\right) = \lambda_{ijk}\epsilon_{ijk}\left[\cos\theta_{ijk} - \cos\theta_{0ijk}\right]^2 \exp\left(\frac{\gamma_{ij}\sigma_{ij}}{r_{ij}-a_{ij}\sigma_{ij}}\right) \exp\left(\frac{\gamma_{ik}\sigma_{ik}}{r_{ik}-a_{ik}\sigma_{ik}}\right)$ is the three-body term. The two body term is similar to *Lennard-Jones Potential* to model the pairwise relationships and the three body term can consider the angles among atom triplets. Similarly, the temperature is changed along with the time to model the environmental variations and a high temperature would also bring a more intense molecular motion.

- *CylinderFlow* is a popular computational fluid dynamics (CFD) simulation dataset, which models the fluid flow around a given cylinder by OpenFoam [122]. It consists of simulation data from modeling an incompressible flow governed by the Navier-Stokes equations. The Reynolds number, denoted by Re, is a dimensionless quantity that characterizes the flow regime of the fluid. It is defined as: $Re = \frac{\rho V D}{\mu}$, where ρ is the density of the fluid, V is the velocity of the fluid relative to the cylinder, D is the diameter of the cylinder, and μ is the dynamic viscosity of the fluid. The transition from laminar to turbulent flow usually occurs at a critical Reynolds number, which depends on the geometry of the cylinder and the properties of the fluid. In general, the flow is more likely to be laminar for small cylinders and viscous fluids, and more likely to be turbulent for large cylinders and low-viscosity fluids. Notably, the initial flow velocity V of the incoming water flow to the cylinder varies cyclically over time, meaning the Reynolds number of the flow field also changes periodically.

- *Airfoil* is generated in a similar manner through simulations of a compressible flow by OpenFoam [122]. The lift coefficient of an airfoil depends on a number of factors, including the angle of attack, the shape of the airfoil, and the Reynolds number of the flow. The angle of attack is the angle between the chord line of the airfoil (the straight line connecting the leading and trailing edges) and the direction of

the incoming flow. The Reynolds number, as mentioned in the previous question, is a dimensionless quantity that characterizes the flow regime of the fluid. It also plays a crucial role in the lift produced by an airfoil, as it determines whether the flow around the airfoil is laminar or turbulent. For laminar flow, the air moves smoothly over the surface of the airfoil, while for turbulent flow, it moves in a chaotic, swirling pattern. The Reynolds number is given by: $Re = \frac{\rho V c}{\mu}$, where c is the chord length of the airfoil, and μ is the viscosity of the fluid. The lift coefficient is typically higher for laminar flow than for turbulent flow, up to a certain point where the flow separates from the airfoil. Notably, in our simulation datasets, the inlet velocity V over the wing also varies cyclically over time.

B.2.2 Details of Baselines

Our proposed method is compared with a range of competing baselines as follows:

- LSTM [49] is a widely recognized approach for sequence prediction problems. It involves three gates, i.e., forget gate, input gate and output gate, enabling the model to acquire knowledge of long-term relationships.

- STGCN [108] is a deep learning approach to handle spatial dependencies and temporal dynamics in complicated spatio-temporal data. It involves a recurrent component and a message passing component for effective analysis of spatio-temporal signals.

- GNS [107] utilizes a graph to represent a physical dynamical system and then utilizes a message passing neural network to explore complicated dynamics and interactions among multiple objects.

- MeshGraphNet [109] characterize each physical system as meshes, followed by graph

neural networks to learn interacting dynamics. Moreover, remeshing techniques are adopted to fit the multi-resolution nature in irregular meshes.

- CG-ODE [119] models both nodes and edges jointly through two groups of ODEs, which can capture the evolution of both objective and interaction in the system.
- TIE [111] attempts to improve the particle-based simulations by decomposing edges into both ends, and introducing abstract nodes to capture global information in the system.
- MP-NODE [110] is an ODE-based approach for homogeneous dynamical systems. It adds neighborhood information into node updating by augmenting the latent dimension.

B.3 POEM

B.3.1 Dataset Details

Table B.1: All of our datasets are listed in detail. The system describes the underlying PDE: cloth, hypere-lastic flow, or a compressible or incompressible Navier-Stokes flow. Simulation data is generated using a solver. In DeformingPlate, there is no time step since it is a quasi-static simulation.

Dataset	Nodes (avg)	System	Type	Solver	Steps	Interval/s
CylinderFLow	1885	Incompressible NS	Eulerian	COMSOL	500	0.01
Airfoil	5233	Compressible NS	Eulerian	SU2	500	0.008
DeformingPlate	1271	Hyper-elasticity	Lagrangian	COMSOL	300	-
FlagSimple	1579	Cloth	Lagrangian	ArcSim	300	0.02

Four physics simulation benchmark datasets are utilized to evaluate our proposed POEM and the compared baselines with details listed in Table B.1.

CylinderFlow simulates the incompressible Navier-Stokes flow of water around a cylinder on a fixed 2D Eulerian mesh generated by COMSOL [123]. The irregular mesh

structure has varying edge lengths in different regions. Each trajectory, containing 1,800 nodes on average, has 500 time steps with an interval 0.01s. Node attributes in the system include mesh position, node type, velocity, and pressure. Node types can be divided into 3 categories in fluid domains, i.e., fluid nodes, wall nodes, and inflow/outflow boundary nodes. We predict the velocity values in both directions, denoted as v_x and v_y, and the pressure field p.

Airfoil simulates the aerodynamics around the cross-section of an airfoil wing for compressible Navier-Stokes flow by SU2 [124]. As the edge lengths of the mesh range between 2×10^{-4}m to 3.5m, the mesh structure is highly irregular. Each trajectory containing 5,200 nodes averagely has 500 time steps with an interval 0.008s. Node attributes include mesh position, node type, velocity, pressure, and density. Besides v_x, v_y and p, we predict the density field d as well.

DeformingPlate is a hyper-elastic plate in the structural mechanical system, deformed by a kinematic actuator, simulated with a quasi-static simulator COMSOL. Each trajectory has 300 time steps with 1,200 nodes averagely. A one-hot vector for each type of node distinguishes actuators from plates in the Lagrangian tetrahedral mesh. Besides, node type, position, and velocity are fed to predict the velocity v_x, v_y and the von-Mises s.

FlagSimple uses a static mesh and ignores collisions to simulate a flag waving in the wind, which is generated by ArcSim [125] for simulating the cloth system. An average trajectory contains 1,500 nodes, which is observed in 300 time steps with the interval 0.02s. The node type distinguishes cloth and obstacle/boundary nodes, and we encode it together with position and velocity to predict the acceleration in both directions, i.e., a_x and a_y. Of note, our four datasets are chosen from four physical domains, i.e., incompressible fluids, compressible fluids, structural mechanics, and cloth, respectively. Each dataset contains 1000 training and 100 validation, and 100 test trajectories.

B.3.2 Details of Baselines

The baselines are the same as what we use in the analysis of CARE.

B.4 GraphSDE

B.4.1 Details of Datasets

GraphSDE is evaluated in four dynamic system datasets, encompassing diverse domains and showcasing the versatility of our approach.

- *COVID-19* [117] used in this study includes COVID-19 data from the Johns Hopkins University, focusing on state-level daily cumulative deaths in the United States and incorporating various dynamic features like confirmed cases, deaths, recoveries, mortality rate, testing rate, and state population. It also integrates mobility data from SafeGraph, capturing people's movements between and within states, to construct an interaction graph and provide an additional dynamic feature for each node. The goal is to predict future trends conditioning on historical data.

- *Radar Map* [1] aims at short-term quantitative precipitation forecasting. It includes real radar reflectivity maps from meteorological observatories, each covering a 101x101 km^2 area around a target site. These maps, collected at fifteen different time spans and four varying heights, offer a comprehensive view of the spatio-temporal patterns in rainfall prediction. We select 30 observation points and the goal is to forecast the radar reflectivity accurately at these spots.

- *Social network* [126] investigates the dynamics of opinion dispersion within social networks, offering insights into the complex interactions that underpin information

[1] from `https://tianchi.aliyun.com/competition/entrance/231596`

spread in societal contexts. It simulates opinion migration over time, featuring 80 nodes and 399 timestamps, with individuals' initial opinions following a uniform distribution in a 2D space. The goal is to predict future opinion migration based on historical opinion propagation.

- *Stock* [127] comprises historical data of 80 stocks in the Nasdaq market, sourced from Yahoo Finance. Each stock's daily record includes key financial metrics such as open, low, high, close, adjusted close, and volume values. The goal is to forecast stock movements, based on historical stock market information.

B.4.2 Details of Baselines

The baselines are the same as what we use in the analysis of HOPE.

Bibliography

[1] K.-C. Tseng, O. Kwon, and L. Tjing, *Time series and neural network forecasts of daily stock prices*, *Investment Management and Financial Innovations* (2012), no. 9, Iss. 1 32–54.

[2] M. Wen, P. Li, L. Zhang, and Y. Chen, *Stock market trend prediction using high-order information of time series*, *Ieee Access* **7** (2019) 28299–28308.

[3] Y. Leu, C.-P. Lee, and Y.-Z. Jou, *A distance-based fuzzy time series model for exchange rates forecasting*, *Expert Systems with Applications* **36** (2009), no. 4 8107–8114.

[4] Y. Aviv, *A time-series framework for supply-chain inventory management*, *Operations Research* **51** (2003), no. 2 210–227.

[5] P. Doganis, E. Aggelogiannaki, and H. Sarimveis, *A combined model predictive control and time series forecasting framework for production-inventory systems*, *International Journal of Production Research* **46** (2008), no. 24 6841–6853.

[6] D. Foreman-Mackey, E. Agol, S. Ambikasaran, and R. Angus, *Fast and scalable gaussian process modeling with applications to astronomical time series*, *The Astronomical Journal* **154** (2017), no. 6 220.

[7] A. Bărbulescu, *Studies on time series applications in environmental sciences*, vol. 103. Springer, 2016.

[8] J. A. Bilmes and C. Bartels, *Graphical model architectures for speech recognition*, *IEEE signal processing magazine* **22** (2005), no. 5 89–100.

[9] J. C. B. Gamboa, *Deep learning for time-series analysis*, *arXiv preprint arXiv:1701.01887* (2017).

[10] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, *arXiv preprint arXiv:1810.04805* (2018).

[11] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et. al.*, *Language models are unsupervised multitask learners*, *OpenAI blog* **1** (2019), no. 8 9.

[12] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et. al.*, *Language models are few-shot learners*, *Advances in neural information processing systems* **33** (2020) 1877–1901.

[13] Z. Gao, G. Lu, P. Yan, and L. Wang, *Retrospective analysis of time series for frame selection in surveillance video summarization*, *Signal, Image and Video Processing* **11** (2017) 581–588.

[14] Y. Zou, R. V. Donner, N. Marwan, J. F. Donges, and J. Kurths, *Complex network approaches to nonlinear time series analysis*, *Physics Reports* **787** (2019) 1–97.

[15] M. Imhoff, M. Bauer, U. Gather, and D. Löhlein, *Time series analysis in intensive care medicine*, tech. rep., Technical Report, 1998.

[16] S. Aydin, *Time series analysis and some applications in medical research*, *Journal of Mathematics and Statistics Studies* **3** (2022), no. 2 31–36.

[17] A. Zeroual, F. Harrou, A. Dairi, and Y. Sun, *Deep learning methods for forecasting covid-19 time-series data: A comparative study*, *Chaos, solitons & fractals* **140** (2020) 110121.

[18] H. Qi, S. Xiao, R. Shi, M. P. Ward, Y. Chen, W. Tu, Q. Su, W. Wang, X. Wang, and Z. Zhang, *Covid-19 transmission in mainland china is associated with temperature and humidity: A time-series analysis*, *Science of the total environment* **728** (2020) 138778.

[19] T. Boehme, A. R. Wallace, and G. P. Harrison, *Applying time series to power flow analysis in networks with high wind penetration*, *IEEE transactions on power systems* **22** (2007), no. 3 951–957.

[20] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[21] J. Contreras, R. Espinola, F. J. Nogales, and A. J. Conejo, *Arima models to predict next-day electricity prices*, *IEEE transactions on power systems* **18** (2003), no. 3 1014–1020.

[22] V. Ş. Ediger and S. Akar, *Arima forecasting of primary energy demand by fuel in turkey*, *Energy policy* **35** (2007), no. 3 1701–1708.

[23] C. C. Holt, *Forecasting seasonals and trends by exponentially weighted moving averages*, *International journal of forecasting* **20** (2004), no. 1 5–10.

[24] R. J. Hyndman, A. B. Koehler, R. D. Snyder, and S. Grose, *A state space framework for automatic forecasting using exponential smoothing methods*, *International Journal of forecasting* **18** (2002), no. 3 439–454.

[25] R. J. Hyndman and Y. Khandakar, *Automatic time series forecasting: the forecast package for r*, *Journal of statistical software* **27** (2008) 1–22.

[26] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice*. OTexts, 2018.

[27] R. B. Cleveland, W. S. Cleveland, J. E. McRae, and I. Terpenning, *Stl: A seasonal-trend decomposition*, *J. Off. Stat* **6** (1990), no. 1 3–73.

[28] P. J. Brockwell and R. A. Davis, *Introduction to time series and forecasting*. Springer, 2002.

[29] X. Wang, K. Smith, and R. Hyndman, *Characteristic-based clustering for time series data*, *Data mining and knowledge Discovery* **13** (2006) 335–364.

[30] C. A. Sims, *Macroeconomics and reality*, *Econometrica: journal of the Econometric Society* (1980) 1–48.

[31] C. A. Sims, *Interpreting the macroeconomic time series facts: The effects of monetary policy*, *European economic review* **36** (1992), no. 5 975–1000.

[32] H. Lütkepohl, *New introduction to multiple time series analysis*. Springer Science & Business Media, 2005.

[33] B. Pfaff, *Analysis of integrated and cointegrated time series with R*. Springer Science & Business Media, 2008.

[34] R. E. Kalman, *A new approach to linear filtering and prediction problems*, .

[35] J. Durbin and S. J. Koopman, *Time series analysis by state space methods oxford university press*, 2012.

[36] M. West and J. Harrison, *Bayesian forecasting and dynamic models*. Springer Science & Business Media, 2006.

[37] G. Petris, S. Petrone, and P. Campagnoli, *Dynamic linear models with R*. Springer Science & Business Media, 2009.

[38] J. F. C. Kingman, *Poisson processes*, vol. 3. Clarendon Press, 1992.

[39] A. G. Hawkes, *Spectra of some self-exciting and mutually exciting point processes*, *Biometrika* **58** (1971), no. 1 83–90.

[40] T. Liniger, *Multivariate hawkes processes*. PhD thesis, ETH Zurich, 2009.

[41] A. Reinhart, *A review of self-exciting spatio-temporal point processes and their applications*, *Statistical Science* **33** (2018), no. 3 299–318.

[42] P. Hartman, *Ordinary differential equations*. SIAM, 2002.

[43] J. C. Butcher, *Numerical methods for ordinary differential equations*. John Wiley & Sons, 2016.

[44] L. C. Evans, *Partial differential equations american mathematical society, Providence, RI* **2** (1998).

[45] K. Itô, *109. stochastic integral*, *Proceedings of the Imperial Academy* **20** (1944), no. 8 519–524.

[46] B. Oksendal, *Stochastic differential equations: an introduction with applications*. Springer Science & Business Media, 2013.

[47] E. Platen and N. Bruti-Liberati, *Numerical solution of stochastic differential equations with jumps in finance*, vol. 64. Springer Science & Business Media, 2010.

[48] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning representations by back-propagating errors*, *nature* **323** (1986), no. 6088 533–536.

[49] S. Hochreiter and J. Schmidhuber, *Long short-term memory*, *Neural computation* **9** (1997), no. 8 1735–1780.

[50] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, *Empirical evaluation of gated recurrent neural networks on sequence modeling*, *arXiv preprint arXiv:1412.3555* (2014).

[51] A. Graves, A.-r. Mohamed, and G. Hinton, *Speech recognition with deep recurrent neural networks*, in *2013 IEEE international conference on acoustics, speech and signal processing*, pp. 6645–6649, Ieee, 2013.

[52] I. Sutskever, O. Vinyals, and Q. V. Le, *Sequence to sequence learning with neural networks*, *Advances in neural information processing systems* **27** (2014).

[53] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, *Learning phrase representations using rnn encoder-decoder for statistical machine translation*, *arXiv preprint arXiv:1406.1078* (2014).

[54] J. Heaton, *Ian goodfellow, yoshua bengio, and aaron courville: Deep learning: The mit press, 2016, 800 pp, isbn: 0262035618*, *Genetic programming and evolvable machines* **19** (2018), no. 1-2 305–307.

[55] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, *Attention is all you need*, *Advances in neural information processing systems* **30** (2017).

[56] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et. al.*, *An image is worth 16x16 words: Transformers for image recognition at scale*, *arXiv preprint arXiv:2010.11929* (2020).

[57] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, *Informer: Beyond efficient transformer for long sequence time-series forecasting*, in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, pp. 11106–11115, 2021.

[58] H. Wu, J. Xu, J. Wang, and M. Long, *Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting*, *Advances in Neural Information Processing Systems* **34** (2021) 22419–22430.

[59] T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin, *Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting*, in *International Conference on Machine Learning*, pp. 27268–27286, PMLR, 2022.

[60] A. Zeng, M. Chen, L. Zhang, and Q. Xu, *Are transformers effective for time series forecasting?*, in *Proceedings of the AAAI conference on artificial intelligence*, vol. 37, pp. 11121–11128, 2023.

[61] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, *Neural ordinary differential equations*, *Advances in neural information processing systems* **31** (2018).

[62] W. Grathwohl, R. T. Chen, J. Bettencourt, I. Sutskever, and D. Duvenaud, *Ffjord: Free-form continuous dynamics for scalable reversible generative models*, *arXiv preprint arXiv:1810.01367* (2018).

[63] E. Dupont, A. Doucet, and Y. W. Teh, *Augmented neural odes*, *Advances in neural information processing systems* **32** (2019).

[64] Y. Rubanova, R. T. Chen, and D. K. Duvenaud, *Latent ordinary differential equations for irregularly-sampled time series*, *Advances in neural information processing systems* **32** (2019).

[65] A. Lou, D. Lim, I. Katsman, L. Huang, Q. Jiang, S. N. Lim, and C. M. De Sa, *Neural manifold ordinary differential equations*, *Advances in Neural Information Processing Systems* **33** (2020) 17548–17558.

[66] C. Finlay, J.-H. Jacobsen, L. Nurbekyan, and A. Oberman, *How to train your neural ode: the world of jacobian and kinetic regularization*, in *International conference on machine learning*, pp. 3154–3164, PMLR, 2020.

[67] P. Kidger, R. T. Chen, and T. J. Lyons, *" hey, that's not an ode": Faster ode adjoints via seminorms.*, in *ICML*, pp. 5443–5452, 2021.

[68] X. Li, T.-K. L. Wong, R. T. Chen, and D. Duvenaud, *Scalable gradients for stochastic differential equations*, in *International Conference on Artificial Intelligence and Statistics*, pp. 3870–3882, PMLR, 2020.

[69] P. Kidger, J. Foster, X. C. Li, and T. Lyons, *Efficient and accurate gradients for neural sdes*, *Advances in Neural Information Processing Systems* **34** (2021) 18747–18761.

[70] P. Kidger, J. Foster, X. Li, and T. J. Lyons, *Neural sdes as infinite-dimensional gans*, in *International conference on machine learning*, pp. 5453–5463, PMLR, 2021.

[71] J. Jia and A. R. Benson, *Neural jump stochastic differential equations*, *Advances in Neural Information Processing Systems* **32** (2019).

[72] P. Kidger, J. Morrill, J. Foster, and T. Lyons, *Neural controlled differential equations for irregular time series*, *Advances in Neural Information Processing Systems* **33** (2020) 6696–6707.

[73] J. Morrill, P. Kidger, L. Yang, and T. Lyons, *Neural controlled differential equations for online prediction tasks*, *arXiv preprint arXiv:2106.11028* (2021).

[74] J. Morrill, C. Salvi, P. Kidger, and J. Foster, *Neural rough differential equations for long time series*, in *International Conference on Machine Learning*, pp. 7829–7838, PMLR, 2021.

[75] J. D. Hamilton, *Time series analysis*. Princeton university press, 2020.

[76] W. W. Wei, *Multivariate time series analysis and applications*. John Wiley & Sons, 2018.

[77] R. H. Shumway, D. S. Stoffer, and D. S. Stoffer, *Time series analysis and its applications*, vol. 3. Springer, 2000.

[78] F. Doshi-Velez and B. Kim, *Towards a rigorous science of interpretable machine learning*, *arXiv preprint arXiv:1702.08608* (2017).

[79] T. J. Lyons, *Differential equations driven by rough signals*, *Revista Matemática Iberoamericana* **14** (1998), no. 2 215–310.

[80] T. Lyons and Z. Qian, *System control and rough paths*. Oxford University Press, 2002.

[81] A. Lejay, *An introduction to rough paths*, in *Séminaire de probabilités XXXVII*, pp. 1–59. Springer, 2003.

[82] M. Gubinelli, *Controlling rough paths*, *Journal of Functional Analysis* **216** (2004), no. 1 86–140.

[83] T. J. Lyons, M. Caruana, and T. Lévy, *Differential equations driven by rough paths*. Springer, 2007.

[84] P. K. Friz and N. B. Victoir, *Multidimensional stochastic processes as rough paths: theory and applications*, vol. 120. Cambridge University Press, 2010.

[85] T. Lyons, *Rough paths, signatures and the modelling of functions on streams*, *arXiv preprint arXiv:1405.4537* (2014).

[86] B. Hambly and T. Lyons, *Uniqueness for the signature of a path of bounded variation and the reduced path group*, *Annals of Mathematics* (2010) 109–167.

[87] C. Reutenauer, *Free lie algebras*, in *Handbook of algebra*, vol. 3, pp. 887–903. Elsevier, 2003.

[88] J. Reizenstein, *Calculation of iterated-integral signatures and log signatures*, *arXiv preprint arXiv:1712.02757* (2017).

[89] J. Reizenstein and B. Graham, *The iisignature library: efficient calculation of iterated-integral signatures and log signatures*, *arXiv preprint arXiv:1802.08252* (2018).

[90] P. Kidger and T. Lyons, *Signatory: differentiable computations of the signature and logsignature transforms, on both cpu and gpu*, *arXiv preprint arXiv:2001.00706* (2020).

[91] M. Lemercier, C. Salvi, T. Damoulas, E. Bonilla, and T. Lyons, *Distribution regression for sequential data*, in *International Conference on Artificial Intelligence and Statistics*, pp. 3754–3762, PMLR, 2021.

[92] Y. Bengio, P. Simard, and P. Frasconi, *Learning long-term dependencies with gradient descent is difficult*, *IEEE transactions on neural networks* **5** (1994), no. 2 157–166.

[93] S. Liao, T. Lyons, W. Yang, and H. Ni, *Learning stochastic differential equations using rnn with log signature features*, *arXiv preprint arXiv:1908.08286* (2019).

[94] J. W. Cooley and J. W. Tukey, *An algorithm for the machine calculation of complex fourier series*, *Mathematics of computation* **19** (1965), no. 90 297–301.

[95] N. R. Lomb, *Least-squares frequency analysis of unequally spaced data*, *Astrophysics and space science* **39** (1976) 447–462.

[96] J. D. Scargle, *Studies in astronomical time series analysis. ii-statistical aspects of spectral analysis of unevenly spaced data*, *Astrophysical Journal, Part 1, vol. 263, Dec. 15, 1982, p. 835-853.* **263** (1982) 835–853.

[97] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[98] E. Lindelöf, *Sur l'application de la méthode des approximations successives aux équations différentielles ordinaires du premier ordre*, *Comptes rendus hebdomadaires des séances de l'Académie des sciences* **116** (1894), no. 3 454–457.

[99] H. Whitney, *Differentiable manifolds*, *Annals of Mathematics* (1936) 645–680.

[100] X. Liu, T. Xiao, S. Si, Q. Cao, S. Kumar, and C.-J. Hsieh, *Neural sde: Stabilizing neural ode networks with stochastic noise*, *arXiv preprint arXiv:1906.02355* (2019).

[101] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, *Generative adversarial networks*, *Communications of the ACM* **63** (2020), no. 11 139–144.

[102] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, *Deep unsupervised learning using nonequilibrium thermodynamics*, in *International conference on machine learning*, pp. 2256–2265, PMLR, 2015.

[103] J. Ho, A. Jain, and P. Abbeel, *Denoising diffusion probabilistic models*, *Advances in neural information processing systems* **33** (2020) 6840–6851.

[104] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, *Score-based generative modeling through stochastic differential equations*, *arXiv preprint arXiv:2011.13456* (2020).

[105] X. Luo, J. Yuan, Z. Huang, H. Jiang, Y. Qin, W. Ju, M. Zhang, and Y. Sun, *Hope: High-order graph ode for modeling interacting dynamics*, in *International Conference on Machine Learning*, pp. 23124–23139, PMLR, 2023.

[106] X. Luo, H. Wang, Z. Huang, H. Jiang, A. Gangan, S. Jiang, and Y. Sun, *Care: Modeling interacting dynamics under temporal environmental variation*, *Advances in Neural Information Processing Systems* **36** (2024).

[107] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. Battaglia, *Learning to simulate complex physics with graph networks*, in *International conference on machine learning*, pp. 8459–8468, PMLR, 2020.

[108] B. Yu, H. Yin, and Z. Zhu, *Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting*, *arXiv preprint arXiv:1709.04875* (2017).

[109] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. W. Battaglia, *Learning mesh-based simulation with graph networks*, *arXiv preprint arXiv:2010.03409* (2020).

[110] J. Gupta, S. Vemprala, and A. Kapoor, *Learning modular simulations for homogeneous systems*, *Advances in Neural Information Processing Systems* **35** (2022) 14852–14864.

[111] Y. Shao, C. C. Loy, and B. Dai, *Transformer with implicit edges for particle-based physics simulation*, in *European Conference on Computer Vision*, pp. 549–564, Springer, 2022.

[112] Y. Liu, T. Hu, H. Zhang, H. Wu, S. Wang, L. Ma, and M. Long, *itransformer: Inverted transformers are effective for time series forecasting*, *arXiv preprint arXiv:2310.06625* (2023).

[113] Y. Zhang and J. Yan, *Crossformer: Transformer utilizing cross-dimension dependency for multivariate time series forecasting*, in *The eleventh international conference on learning representations*, 2022.

[114] L. Han, H.-J. Ye, and D.-C. Zhan, *The capacity and robustness trade-off: Revisiting the channel independent strategy for multivariate time series forecasting*, *arXiv preprint arXiv:2304.05206* (2023).

[115] J. Neyman and E. S. Pearson, *Ix. on the problem of the most efficient tests of statistical hypotheses*, *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character* **231** (1933), no. 694-706 289–337.

[116] A. Camuto, M. Willetts, S. Roberts, C. Holmes, and T. Rainforth, *Towards a theoretical understanding of the robustness of variational autoencoders*, in *International Conference on Artificial Intelligence and Statistics*, pp. 3565–3573, PMLR, 2021.

[117] E. Dong, H. Du, and L. Gardner, *An interactive web-based dashboard to track covid-19 in real time*, *The Lancet infectious diseases* **20** (2020), no. 5 533–534.

[118] D. J. Berndt and J. Clifford, *Using dynamic time warping to find patterns in time series*, in *Proceedings of the 3rd international conference on knowledge discovery and data mining*, pp. 359–370, 1994.

[119] Z. Huang, Y. Sun, and W. Wang, *Coupled graph ode for learning interacting system dynamics*, in *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, pp. 705–715, 2021.

[120] H. Xia, V. Suliafu, H. Ji, T. Nguyen, A. Bertozzi, S. Osher, and B. Wang, *Heavy ball neural ordinary differential equations*, *Advances in Neural Information Processing Systems* **34** (2021) 18646–18659.

[121] F. Li, J. Feng, H. Yan, G. Jin, F. Yang, F. Sun, D. Jin, and Y. Li, *Dynamic graph convolutional recurrent network for traffic prediction: Benchmark and solution*, *ACM Transactions on Knowledge Discovery from Data* **17** (2023), no. 1 1–21.

[122] H. Jasak, *Openfoam: Open source cfd in research and industry*, *International Journal of Naval Architecture and Ocean Engineering* **1** (2009), no. 2 89–94.

[123] C. Multiphysics, *Introduction to comsol multiphysics®*, *COMSOL Multiphysics, Burlington, MA, accessed Feb* **9** (1998), no. 2018 32.

[124] T. D. Economon, F. Palacios, S. R. Copeland, T. W. Lukaczyk, and J. J. Alonso, *Su2: An open-source suite for multiphysics simulation and design*, *Aiaa Journal* **54** (2016), no. 3 828–846.

[125] R. Narain, A. Samii, and J. F. O'brien, *Adaptive anisotropic remeshing for cloth simulation*, *ACM transactions on graphics (TOG)* **31** (2012), no. 6 1–10.

[126] Y. Gu, Y. Sun, and J. Gao, *The co-evolution model for social network evolving and opinion migration*, in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 175–184, 2017.

[127] A. Jafari and S. Haratizadeh, *Gcnet: graph-based prediction of stock price movement using graph convolutional network*, *Engineering Applications of Artificial Intelligence* **116** (2022) 105452.