Multivariate Time Series Forecasting With Dynamic Graph Neural ODEs

Ming Jin, Yu Zheng, Yuan-Fang Li, Siheng Chen, Bin Yang, and Shirui Pan

Abstract—Multivariate time series forecasting has long received significant attention in real-world applications, such as energy consumption and traffic prediction. While recent methods demonstrate good forecasting abilities, they have three fundamental limitations. (i) Discrete neural architectures: Interlacing individually parameterized spatial and temporal blocks to encode rich underlying patterns leads to discontinuous latent state trajectories and higher forecasting numerical errors. (ii) High complexity: Discrete approaches complicate models with dedicated designs and redundant parameters, leading to higher computational and memory overheads. (iii) Reliance on graph priors: Relying on predefined static graph structures limits their effectiveness and practicability in real-world applications. In this paper, we address all the above limitations by proposing a continuous model to forecast multivariate time series with dynamic graph neural ordinary differential equations (MTGODE). Specifically, we first abstract multivariate time series into dynamic graphs with time-evolving node features and unknown graph structures. Then, we design and solve a neural ODE to complement missing graph topologies and unify both spatial and temporal message passing, allowing deeper graph propagation and fine-grained temporal information aggregation to characterize stable and precise latent spatial-temporal dynamics. Our experiments demonstrate the superiorities of MTGODE from various perspectives on five time series benchmark datasets.

1 INTRODUCTION

Time series data plays a vital role in shaping modern societies and has long been studied across multiple fields in science and engineering, such as energy grid balancing [1], climate studies [2], and traffic volume forecasting [3]. Among these applications and given a sensor network, the multivariate time series data can be interpreted as the combination of recorded univariate time series on each sensor, which can be interconnected and mutually influenced. For example, the rise in daily average temperature may lead to an increase in traffic volume on coastal roads. Therefore, multivariate time series forecasting largely depends on modeling the underlying spatial-temporal correlations, which directly affects the reliability of those above and many other real-world applications.

Nevertheless, this task is not easy because it is principally challenging to effectively and efficiently model the underlying complex spatial-temporal dependencies on multivariate time series. While earlier methods are based on statistical models [4], [5], [6], recent works take deep learning-based approaches, demonstrating better capabilities to capture non-linear temporal and spatial patterns. Although recurrent neural networks (RNNs) [7] have been widely adopted in modeling multivariate time series, vanilla RNNs and their variants [9], [10], [11] also fail to exploit the dynamic interdependencies among variables. To address the above limitations, LSTNet [12] adopts 1D convolution neural network (CNN) and two RNN variants to capture short-term local variable dependencies and long-term temporal patterns. TPA-LSTM [13] first processes the input sequences via an RNN and then leverages multiple 1D convolution filters and a scoring function to capture both temporal and spatial correlations. HyDCNN [14], on the other hand, designs a CNN-based model to capture rich spatial and temporal patterns simultaneously. However, these methods do not explicitly model the pairwise dependencies between variables, limiting their effectiveness in forecasting multivariate time series.

Recently, techniques based on graph neural networks (GNNs) [15] have demonstrated great potential in modeling the spatial and temporal interdependencies simultaneously among multiple time series over time. As a specific data format, graphs can naturally be adopted to describe the inter-connections between entities. In the context of multivariate time series forecasting, the essence of these methods is...
MTGODE allows modeling the fully continuous latent spatial-temporal dynamics. Compared with existing methods, MTGODE generates the discrete graph propagation with an ODE, its temporal aggregation remains discrete. In contrast, MTGODE allows modeling the fully continuous latent spatial-temporal dynamics.

Fig. 1. The latent state trajectories of different forecasting methods when encoding historical observations. The vertical dash lines denote temporal aggregation steps, and the three colored solid lines are latent state trajectories of different variables. Among these methods, GRU has a sequence of transformations (i.e., graph propagation) after each temporal aggregation step. Although STGODE generates the discrete graph propagation with an ODE, its temporal aggregation remains discrete. In contrast, MTGODE allows modeling the fully continuous latent spatial-temporal dynamics.

predicting future node features with the help of historical observations and predefined graph structures, where nodes, node features, and static edges are variables, univariate time series, and the prior knowledge to describe stable relationships between variables (e.g., metro networks), respectively.

For instance, DCRNN [16] proposes a bidirectional graph random walks-based gated recurrent unit to model spatial and temporal dependencies, STGCN [3] intersects graph and temporal convolutions to learn on multivariate time series data, and GMAN [17] designs a spatial-temporal block to do similar things by composing two attention mechanisms on the graph and temporal spaces. Although these methods demonstrate competitive performances, it remains difficult for them to accurately model arbitrary multivariate time series based on the following challenges:

- **Challenge 1: Discrete Neural Architectures.** Instead of parameterizing the continuous dynamics of latent states, the existing works on modeling multivariate time series are based on entirely or partially discrete neural architectures, resulting in discontinuous state trajectories in modeling latent spatial-temporal dynamics, which is shown to be less effective [18] and thus hinders downstream tasks in terms of the forecasting precision. Fig. 1 plots the latent state trajectories of four typical methods when learning on multivariate time series data. Specifically, most of these approaches fail to define a vector field to characterize the fully continuous latent spatial-temporal dynamics except for our method, which demonstrates significantly better downstream forecasting performance (Tables 3 and 4). Another limitation of discrete neural architectures is the shallow graph propagation in most GNN-based approaches, such as STGCN [3] and MTGNN [19], due to the challenge of over-smoothing, which prevents them from considering the spatial correlations from farther neighbors and further limiting their forecasting abilities.

- **Challenge 2: High Complexity.** Discretely stacking individually parameterized spatial and temporal modules, e.g., in [3] and [19], not only results in discontinuous latent state trajectories but also complicates models with dedicated designs (e.g., parameterized residual and skip connections) and redundant trainable parameters, leading to computational and memory inefficient. We theoretically and empirically justify this in Sections 4.4, 5.4, and 5.7.

- **Challenge 3: Rely on Graph Priors.** Plenty of existing GNN-based forecasting models, e.g., [3], [17], [20], and [21], require prior knowledge of graph structures (i.e., stable interconnections between variables). However, such knowledge is typically unknown in most cases, hindering their applications in broader real-world applications.

Although some recent works aim to address the above limitations, none of these methods can solve them all. For example, STGODE [21] proposes an ODE network to characterize the continuous propagation on predefined graphs, but its temporal aggregation process remains discrete. Thus, it faces all the above challenges. Other methods, such as GTS [22] and MTGNN [19], get rid of the predefined graph structures, but they leave the first and second limitations unsolved. While a recently proposed method, STGNCDE [23], addresses the first and third challenges with Neural Controlled Differential Equations (NCDEs) [24] and graph structure learning, it remains complex and less effective when modeling long input series because of its recursive nature and interpolating preprocessing.

In this paper, we resolve all the above challenges by proposing a novel method to forecast Multivariate Time series with dynamic Graph neural Ordinary Differential Equations (MTGODE for abbreviation). Compared with existing works, our theme allows explicitly encoding the fully continuous spatial-temporal dynamics of arbitrary multivariate time series in the latent space, which benefits various downstream forecasting tasks by a large margin. Specifically, we abstract input series as dynamic graphs with time-evolving node features and unknown graph structures. To complement and learn from the missing interdependencies between nodes (i.e., variables), we propose a continuous graph propagation mechanism together with a graph structure learning schema to partially and wholly address the first and third challenges, which significantly alleviates the over-smoothing issue in GNNs and thus allows deeper continuous propagation on dynamically self-extracted graph structures to capture long-range spatial correlations between time series. To encode rich temporal information and completely resolve the first challenge, we propose a continuous temporal aggregation mechanism to parameterize the derivative of latent states instead of themselves, allowing fine-grained temporal patterns to be extracted and aggregated precisely. It is worth noting that this mechanism also tactfully addresses the second challenge by eliminating redundant computations and 1. Code is available at https://github.com/GRAND-Lab/MTGODE
disentangling the ties between aggregation depth and memory bottleneck in discrete formulations, thus can provide more accurate modeling of latent temporal dynamics than discrete methods with limited computational budgets. In MTGODE, we elegantly couple two mechanisms and provide a simpler framework than most GNN-based forecasting pioneers to learn from and forecast multivariate time series that are both more effective and efficient, thus resolving the three aforementioned challenges. We summarize our contributions as follows:

- To the best of our knowledge, this is the first work to learn fully continuous latent spatial-temporal dynamics of arbitrary multivariate time series by unifying spatial and temporal message passing with two coupled ODEs and a more concise model design.
- We propose a spatial ODE together with a graph learning schema to learn continuous long-range spatial dynamics between time series, which alleviates the reliance on static graph priors and the common over-smoothing problem in GNNs.
- We propose a temporal neural ODE by generalizing canonical temporal convolutions to learn the continuous fine-grained temporal dynamics of time series, resulting in a powerful and efficient forecasting model with the proposed spatial ODE.
- We conduct extensive experiments to demonstrate the effectiveness and efficiency of the proposed method, showing better application prospects.

We organize the rest of the paper as follows: Section 2 reviews the related work. Section 3 provides the problem definition and notations. Section 4 presents the proposed method and algorithms. In Sections 5 and 6, we discuss the experimental results and conclude the paper.

## 2 RELATED WORK

This work is closely related to multivariate time series forecasting, graph neural networks, and neural ordinary differential equations. We briefly review related representative works in this section.

### 2.1 Multivariate Time Series Forecasting

Multivariate time series forecasting has long been a widely studied subject, where existing approaches are in two categories: Statistical and deep learning-based. For the former branch of methods, auto-regressive (AR) models linearly predict future changes in a time series based on historical observations. Vector auto-regressive (VAR) approaches further extend AR by exploring the interdependencies between multiple time series. ARIMA [25], on the other hand, integrates the ideas of AR and moving average (MA). On top of this, VARIMA [26] generates ARIMA to operate on multivariate time series. Although statistical models are widely applied in real-world applications because of their interpretability and simplicity, they only explore linear relationships and make strong assumptions of stationary processes. In contrast, recent deep learning-based methods are free from these limitations and demonstrate better performances. LSTNet [12] and TPA-LSTM [13] are two models that propose to learn discrete temporal dynamics and local spatial correlations between time series via RNNs and CNNs. Recently, to address the parallelization issue in RNNs, methods built on CNNs or Transformer [27] demonstrate a better efficiency and forecasting ability. For example, HyDCNN [14] adopts position-dilated CNNS to model both spatial and temporal information, and Informer [28] does the same things with a proposed variant of vanilla Transformer. However, all those methods have not explicitly modeled the pairwise dependencies between variables, limiting their effectiveness in forecasting multivariate time series. In this work, we first abstract multivariate time series as dynamic graphs with time-evolving node features and unknown graph structures, and then complement and learn from the missing interdependencies between nodes with the proposed continuous graph propagation mechanism, which allows MTGODE better to capture the pairwise dependencies between nodes (i.e., variables) explicitly in arbitrary multivariate time series.

### 2.2 Graph Neural Networks

Graphs are ubiquitous in the real world, and GNNs are designed to incorporate attributive and topological information to learn expressive node-level or graph-level representations [29], [30], where spatial correlations between nodes are explicitly modeled by passing messages from nodes’ neighbors to nodes themselves. Recently, several works have emerged to tackle the traffic forecasting problem with GNN-based models [3], [8], [16], [17], [19], [20], [21], [22], [23]. Given an input multivariate time series and a predefined graph structure, we can characterize the static relationships between variables (i.e., nodes), they typically adopt graph convolutions to capture local spatial dependencies and use RNNs [16], [22], or 1D convolutions [3], [8], [19] to model temporal dynamics. Although minor works exist to alleviate the reliance on graph priors [19], [22], [23] or conduct deeper graph propagation [21] to capture long-range spatial dependencies, they fail to completely address all three above challenges to effectively and efficiently learn stable and precise spatial-temporal dynamics on arbitrary multivariate time series data in the latent space. To bridge the gaps, we propose a simpler model by elegantly coupling two proposed continuous mechanisms, demonstrating significantly better effectiveness and efficiency.

### 2.3 Neural Ordinary Differential Equations

Chen et al. [18] introduced a new paradigm of continuous-time models by generalizing discrete deep neural networks. Taking a $L$-layer residual network as an example, it can be formulated as follows:

$$
\begin{align*}
\dot{H}_{i,t+1} &= H_i + f(H_i, \Theta_i), \\
H_{i,t+1} &= H_L.
\end{align*}
$$

If we insert more layers and take smaller integration steps, then we can directly parameterize and approximate the continuous evolution of latent states, which forms the basic idea of Neural Ordinary Differential Equations (NODEs)

$$
\begin{align*}
\frac{dH(t)}{dt} &= f(H(t), \Theta), \\
H_{\text{out}} &= \text{ODESolve}(H(0), t_0, t_1, \Theta).
\end{align*}
$$
In the following, we omit \( t_0 \) in the above equation for simplicity if \( t_0 = 0 \). Recently, NODEs have been adopted in some research fields, such as graph neural networks [31], [32] and traffic forecasting [21]. Specifically, as the only ODE-based method for traffic forecasting, STGODE [21] merely considers the continuous graph propagation on predefined static graph structures without modeling the continuous temporal dynamics. Our approach distinguishes from it in two important aspects. First, we propose a novel continuous temporal aggregation mechanism coupled with a simplified continuous graph propagation process to learn more expressive latent spatial-temporal dynamics efficiently in a fully continuous manner. Second, our method eliminates the reliance on predefined graph structures. On the other hand, as an extension of NODEs, Neural Controlled Differential Equations (NCDEs) [24] emerges as a continuous generalization of RNNs to learn on time series data naturally. A recently proposed method, STG-NCDE [23], further extends this idea to model traffic data with two different NCDEs to severally model temporal and spatial dependencies, showing good forecasting results. Although STG-NCDE learns continuous latent dynamics without relying on predefined graph priors, it cannot efficiently and efficiently handle long input series due to its recursive nature like in RNNs and resource-intensive interpolating preprocessing. In contrast, our method is free from this issue with a non-recursive backbone process and a light preprocessing module. Thus, compared with the above two methods, \( \text{MTGODE} \) can efficiently forecast arbitrary multivariate time series with more competitive performance.

### 3 Problem Definition

In this section, we introduce the problem of representation learning on multivariate time series with two commonly adopted evaluation protocols, i.e., single-step and multi-step forecasting. Specifically, the bold uppercase and lowercase letters denote matrices and vectors. We summarize all important notations in Table 1.

Let \( \mathbf{X} \in \mathbb{R}^{N \times D \times S} \) denote a multivariate time series with \( N \) variables, \( D \) feature dimensions, and \( S \) time steps in total for training. Specifically, we define \( \mathbf{X}^t \in \mathbb{R}^{D \times S} \) as the \( t \)th time series for all features and time steps, and \( \mathbf{X}_t \in \mathbb{R}^{N \times D} \) as the \( t \)th time step for all features and features. Given a sequence of \( T \) historical observations \( \mathbf{X}^t_{t+1:T} \in \mathbb{R}^{N \times D \times T} \), our objective is to learn a spatial-temporal encoder \( f(\cdot) : \mathbb{R}^{N \times D \times T} \rightarrow \mathbb{R}^{N \times D'} \), where the learned representation \( \mathbf{h}_{out} = f(\mathbf{X}_{t+1:T}) \) can be used in various downstream tasks, such as the single-step and multi-step forecasting of future observations in Section 5. Formally, given a loss function \( \ell(\cdot) \) and for each valid time step \( t \), we formulate the problem of multivariate time series forecasting as follows:

\[
    f^*, g^* = \arg\min_{f, g} \sum_{t} \ell(g(f(\mathbf{X}^t_{t+1:T})), Y_t),
\]

where \( f^*(\cdot) \) and \( g^*(\cdot) \) represent the encoder and decoder with learned optimal parameters. Specifically, we let \( Y = \mathbf{X}_{t+T:H} \in \mathbb{R}^{N \times D \times 1} \) for single-step forecasting, and \( Y = \mathbf{X}_{t+T+1:T+H} \in \mathbb{R}^{N \times D \times H} \) for multi-step forecasting. \( H \) represents a specific forecasting horizon.

### 4 Methodology

In this section, we present the overall framework and detailed designs of the proposed \( \text{MTGODE} \) method. As shown in Fig. 2, our method mainly consists of three main components, namely the data preprocessing, continuous spatial-temporal encoding, and downstream task decoding. Given a sequence of multivariate time series, we first map it to the latent space with a \( 1 \times 1 \) convolution kernel and acquire the underlying topological structure dynamically with a graph constructor. Then, to extract and encode rich interlaced spatial-temporal patterns of input series, we propose two elegantly coupled continuous processes, i.e., continuous graph propagation (CGP) and continuous temporal aggregation (CTA). Different from [3], [19], and [21], our former process with the graph structure learning not only enables the model to complement and learn from the missing interdependencies between time series but also allows the continuous and further message passing on dynamically self-extracted graph structures to learn continuous long-range spatial dynamics. To effectively extract and aggregate fine-grained temporal patterns, we surrogate discrete temporal convolutions with the proposed CTA process, which defines a vector field to drive and model the underlying latent temporal dynamics continuously. Specifically, we parameterize the derivative of latent states instead of themselves, providing more accurate modeling of complex temporal dependencies, eliminating redundant computations (e.g., parameterized residual and skip connections), and disentangling the ties between

---

**Table 1**

| Symbols                  | Description                                           |
|--------------------------|-------------------------------------------------------|
| \( \mathbf{X} \in \mathbb{R}^{N \times D \times S} \) | A given multivariate time series data sampled from \( \mathbf{X} \) |
| \( \mathbf{X}^t_{t+1:T} \in \mathbb{R}^{N \times D \times T} \) | A sequence of \( T \) historical observations |
| \( \mathbf{A} \in \mathbb{R}^{N \times N} \)            | A learned adjacency matrix                             |
| \( \mathbf{H}^t \in \mathbb{R}^{N \times D' \times Q} \) | The latent state of discrete graph propagation at \( t \)th layer |
| \( \mathbf{H}^G(t) \in \mathbb{R}^{N \times D' \times Q} \) | The intermediate latent state of CGP process at \( t \) |
| \( \mathbf{H}^T(t) \in \mathbb{R}^{N \times D' \times R} \) | The intermediate latent state of CTA process at \( t \) |
| \( \mathbf{H}_{out} \in \mathbb{R}^{N \times D' \times Q} \) | The learned spatial representation of the graph module |
| \( \mathbf{H}_{out} \in \mathbb{R}^{N \times D' \times R} \) | The learned temporal representation of the temporal module |
| \( \Theta, \Phi, \Gamma \) | The trainable parameters of \( \text{MTGODE} \) |
| \( N, S, D \)            | The number of variables, length, and feature dimensions of \( \mathbf{X} \) |
| \( T, H \)               | Input length and forecasting horizon                  |
| \( D' \)                 | The output dimensions of \( \mathbf{H}_{out} \)         |
| \( K, L \)               | The number of layers of discrete graph propagation and temporal aggregation |
| \( T_{cgp}, \Delta t_{cgp} \) | The integration time and step size when defining a CGP process |
| \( T_{cta}, \Delta t_{cta} \) | The integration time and step size when defining a CTA process |
aggregation depth and memory bottleneck in discrete formulations, thus resulting in better model effectiveness and efficiency. In MTGODE, the proposed continuous spatial and temporal message passing mechanisms are elegantly unified; thus, our approach can effectively and efficiently model complex latent spatial-temporal dynamics of arbitrary multivariate time series in a fully continuous manner. Finally, given the learned representations of historical observations, we can conduct various forecasting tasks by employing different downstream decoders, e.g., the single-step forecasting in the rightmost part of Fig. 2.

In the rest of this section, we introduce the proposed CGP and CTA processes in Sections 4.1 and 4.2. In Section 4.3, we theoretically analyse our method by comparing with their discrete variants. The details of model training and algorithms are discussed in Section 4.4.

4.1 Continuous Graph Propagation

In MTGODE, we essentially integrate temporal aggregation and graph propagation processes to capture rich spatial-temporal patterns from historical observations. Specifically, at each temporal aggregation step and for simplicity, the spatial dependencies between time series can be characterized by the combination of feature propagation and linear transformation on a specific graph snapshot with the feature matrix $H^G_0 \in \mathbb{R}^{N \times D}$, which is the latent states at a specific aggregation step (see Sections 4.2 and 4.4 for details). In a nutshell, given an adjacency matrix $A \in \mathbb{R}^{N \times N}$ and initial states $H^G_0$, a discrete formulation of the $K$-hop graph propagation is defined as [33]

$$
\begin{align*}
    H^G_{k+1} &= \hat{A} H^G_k, \quad k \in \{0, \ldots, K - 1\}, \\
    H^G_{\text{out}} &= H^G_K \Phi,
\end{align*}
$$

where $\hat{A}$ denotes the normalized adjacency matrix, $H^G_{\text{out}} \in \mathbb{R}^{N \times D_{\text{out}}}$ is the output representation, and $\Phi \in \mathbb{R}^{D \times D_{\text{out}}}$ is a trainable parameter matrix. In practice, we define tensor multiplication in feature propagation with the Einstein summation in the above equation to sum the element products along specific dimensions. This is because the feature propagation only operates on the first two dimensions of latent states without aggregating information along the time axis (with the sequence length $Q$).

Compared with GCN [34], Equation (4) eliminates the redundant nonlinearities and further decouples the feature propagation and transformation steps, resulting in a simpler and more efficient model while maintaining comparable accuracy. However, this discrete formulation is error-prone and vulnerable to over-smoothing when conducting a deep propagation on graphs. The underlying cause of these two problems in Equation (4) can be uncovered by decomposing the propagation depth $K$ into the combination of integration time $T_{\text{cgp}}$ and step size $\Delta t_{\text{cgp}}$, i.e., $K = T_{\text{cgp}} / \Delta t_{\text{cgp}}$. From the perspective of a continuous process, a selected $T_{\text{cgp}}$ and $\Delta t_{\text{cgp}}$ control the number of function evaluations, which is equivalent to describing how many times feature propagation is executed, a.k.a. the propagation depth $K$ in the discrete formulation. Therefore, considering a case where a fixed integration time and smaller step size are applied, we can naturally have the following transformation with propagation steps $k \in \{0, \ldots, K - 1\}$ being replaced by a continuous variable $t \in \mathbb{R}^+$:

$$
H^G(t + \Delta t_{\text{cgp}}) = H^G(t) + \Delta t_{\text{cgp}} (\hat{A} - I_N) H^G(t) = \left(1 - \Delta t_{\text{cgp}} I_N + \Delta t_{\text{cgp}} \hat{A}\right) H^G(t). \tag{5}
$$

On this basis, we can find that Equation (4) rigidly ties the propagation depth and integration time by enforcing $\Delta t_{\text{cgp}} = 1$ (i.e., the above equation degrades to Equation (4) when step size $\Delta t_{\text{cgp}} = 1$). If so, letting $K = T_{\text{cgp}} \rightarrow \infty$ not only makes the graph Laplacian eigenvalues in a discrete propagation tend to zeros (see Appendix A, available online) but also leads to infinite numerical errors (see Appendix B), available in the online supplemental material, which prevents the model from accurately capturing long-range spatial dependencies. In this work, inspired by [32], we disentangle the coupling between $K$ and $T_{\text{cgp}}$, which alleviates the aforementioned problems by avoiding $T_{\text{cgp}} \rightarrow \infty$. We provide detailed theoretical justifications in Section 4.3. In MTGODE, we generalize Equation (4) with its continuous formulation in the following proposition based on Equation (5), which allows fine-grained and long-range spatial dependencies between time series to be captured.

**Proposition 1.** The continuous dynamics of simplified graph propagation described in Equation (4) admits the following ODE:

$$
\frac{dH^G(t)}{dt} = (\hat{A} - I_N) H^G(t), \tag{6}
$$

Fig. 2. The overall framework of MTGODE. Given a sequence of historical observations, we first map them to the latent space and learn an associated graph structure. Then, the continuous dynamics of spatial and temporal clues are modeled by coupling two ODEs from different perspectives, namely the continuous temporal aggregation (CTA) and continuous graph propagation (CGP) in the middle part. Finally, the learned representations can be used in various forecasting tasks, such as the plotted single-step forecasting in the rightmost part.
where the initial state $H^G(0) = H^G_0$. Specifically, $H^G_0$ is the intermediate state of the continuous temporal aggregation process (Section 4.2) as mentioned before.

To further reduce numerical errors, we propose an attentive transformation to replace the linear mapping in Equation (4), which integrates not only the final but also the initial and selected intermediate states as the output of graph propagation

$$
\begin{align*}
H^G(t_i) &= \text{ODESolve}(H^G(0), \frac{dH^G(t)}{dt}, t_i), \\
H^G_{\text{out}} &= \sum_{t_i} H^G(t_i) \Phi_{t_i}, \quad t_i \in [0, T_{\text{agg}}],
\end{align*}
$$

(7)

where $\text{ODESolve}(\cdot)$ can be any black-box ODE solver introduced in [18]. Specifically, $H^G(t_i)$ denotes the selected intermediate states of a CGP process, and we only take $t_i$ that is divisible by $\Delta_{\text{agg}}$ for simplicity in practice.

**Dynamic Graph Structure Learning.** In Equation (6), it remains unknown how the graph adjacency matrix $A$ is constructed. To address the third challenge and handle multivariate time series without graph priors (e.g., unknown $A$), we adopt a direct optimization approach to learn dynamic graph structures together with the entire model, where node connections evolve with model training. Specifically, for a sequence of historical observations, the underlying adjacency matrix $A$ is dynamically optimized as training progresses to learn to describe the stable interdependencies between variables

$$
\begin{align*}
M^1 &= \tanh(\beta E^1T^1_{ge}), \\
M^2 &= \text{ReLU} \left( \tanh(\beta (M^1M^T_{ij} - M^1M^T_{ij})) \right),
\end{align*}
$$

(8)

where $M^1, M^2 \in \mathbb{R}^{N \times d}$ are described by two neural networks with randomly initialized embedding matrices $E^1, E^2 \in \mathbb{R}^{N \times d}$ and trainable parameters $T^1_{ge}, T^2_{ge} \in \mathbb{R}^{d \times d}$. $\beta$ is a hyperparameter to adjust the activation saturation rate. The learned graph structure is made sparse to reduce the computational cost and is supposed to be uni-directional because changes in a time series are likely to unidirectionally lead to fluctuations in other series [19].

### 4.2 Continuous Temporal Aggregation

Solving the spatial ODE in Equation (7) only allows capturing the spatial dependencies between time series at a certain time step. To learn from the rich temporal information, we treat our spatial ODE as an interior process of the proposed temporal neural ODE, which allows modeling to model precise and stable dynamics of multivariate time series from both spatial and temporal perspectives.

We first introduce the composition of temporal neural ODE to characterize fine-grained and accurate temporal dependencies. Given the shortcomings of RNNs, such as time-consuming iteration and gradient explosion [8], we may stack multiple residual convolution blocks to extract and aggregate temporal patterns in a non-receptive manner

$$
H^T_{t+1} = T(H^T_t, Q_{t+1}) + \text{TCN}(H^T_t, \Theta), \quad l \in \{0, \ldots, L-1\}
$$

(9)

where $\text{TCN}(\cdot, \Theta)$ is an individually parameterized temporal convolution layer, $T(H^T_t, Q_{t+1})$ denotes a truncate function to take only the last $Q_{t+1}$ elements in $H^T_t$ along its last dimension, and $H^T_t \in \mathbb{R}^{N \times d \times Q_t}$ is the output of the $l$th layer with the sequence length $Q_t$. In this formulation, the last dimension of the residual input $H^T_t$ has to be truncated to $Q_{t+1}$ before adding to its transformation because the length of latent representations shrinks gradually after each aggregation step, i.e., $Q_{t+1} = Q_t - r^l \times (k-1)$ and $Q_t = R - r \times (k - 1)$. Specifically, we define $H^T_0 \in \mathbb{R}^{N \times d \times R}$ as the initial state, $r, k, R$ are dilation factor, kernel size, and model receptive field. In practice, we assume $R > T$ to losslessly encode all historical observations, where $R = L(k - 1) + 1$ when $r = 1$, and $R = 1 + (k - 1)(r^l - 1)/(r - 1)$ when $r > 1$.

However, the discrete formulation in Equation (9) suffers from two main limitations. First, it fails to model the fine-grained and accurate temporal dynamics with a fixed large step size in numerical integration, i.e., $\Delta_{\text{agg}} = 1$, which breaks the continuity of the latent state trajectories. Second, it parameterizes convolution layers individually, which has a large number of trainable parameters and relies on dedicated model designs to avoid the gradient vanishing issue and ensure convergence [8], [19], resulting in high computational and memory overheads. Thus, we apply a similar idea to disentangle the ties between aggregation depth $L$ and integration time $T_{\text{tra}}$ by letting $L = T_{\text{tra}}/\Delta_{\text{tra}}$. In such a way, given a desired terminate time $T_{\text{tra}}$ and initial state $H^T_0$, we can characterize the entire continuous temporal aggregation process with a single set of parameters $\Theta$ by letting $\Delta_{\text{tra}} \rightarrow 0$

$$
\begin{align*}
\bar{H}^T_{\text{out}} &= H^T_0 + \int_0^{T_{\text{tra}}} \mathcal{P}(\text{TCN}(H^T_t, \Theta), R) \, dt, \\
H^T_{\text{out}} &= H^T_{\text{out}}[\ldots, -1].
\end{align*}
$$

(10)

To achieve this, we design a simple zero-padding trick to ensure the invariance of latent state dimensions during transformations, where the length (i.e., the last dimension) of latent states are left zero-padded to $R$ with a padding function $\mathcal{P}(\cdot)$ after each step of aggregation. Although the padding is applied, the length of informative parts of latent states shrinks gradually to one after the temporal aggregation (Fig. 2), as same as in temporal convolution networks. Thus, we take $H^T_{\text{out}}[\ldots, -1]$ as the output of the proposed CTA process in Equation (10). On this basis, we have the second proposition defined as follows:

**Proposition 2.** The temporal aggregation process described in Equation (9) is a discretization of the following ODE

$$
\frac{dH^T(t)}{dt} = \mathcal{P}(\text{TCN}(H^T(t), t, \Theta), R),
$$

(11)

with the initial state $H^T(0) = H^T_0$, which is obtained by mapping the input series to the latent space with a separate convolution layer parameterized by $\Theta_{tra}$, i.e., $H^T_0 = \text{CONV}_1(X_{t=0} + f_{\text{g0}}(H^T(t), \Theta_{g0}))$. We denote this mapping as the start convolution in Fig. 2.

Regarding the design of $\text{TCN}(\cdot, \Theta)$ in the above proposition, we adopt a gating mechanism to control the amount of information flows at each integration step

$$
\text{TCN}(H^T(t), \Theta) = fc(H^T(t), \Theta_f) \odot f_g(H^T(t), \Theta_g),
$$

(12)
where $\odot$ denotes the element-wise product. $f_c(\cdot, \Theta_c)$ and $f_g(\cdot, \Theta_g)$ are filtering and gating convolutions that share similar network structures but with different parameters and nonlinearities

$$
\begin{align*}
    f_c(H^T(t), \Theta_c) &= \tanh(W_{tc}^{\Theta_c} \odot H^T(t) + b_{tc}^{\Theta_c}), \\
    f_g(H^T(t), \Theta_g) &= \sigma(W_{tg}^{\Theta_g} \odot H^T(t) + b_{tg}^{\Theta_g}),
\end{align*}
$$

where $\sigma(\cdot)$ represents the sigmoid activation, and $\cdot^d$ denotes the convolution operation with an expandable dilation defined by $\delta = \lfloor r/\Delta t_{eda} \rfloor$. Specifically, $\lfloor \cdot \rfloor$ denotes the floor operation, which outputs the largest integer less than or equal to the input. In practice, adopting a single kernel size is less effective in exploring multi-granularity temporal patterns. Thus, inspired by [19], we equip $f_c(\cdot, \Theta_c)$ and $f_g(\cdot, \Theta_g)$ with multiple convolutions with different kernel widths $m_i$, i.e., $f_c(\cdot, \Theta_{c^m})$ and $f_g(\cdot, \Theta_{g^m})$. Since most of the time series data have inherent periods (e.g., 7, 14, 24, 28, and 30), letting kernel width in set $\{2, 3, 6, 7\}$ makes the aforementioned periods can be fully covered.

Algorithm 1. The Training Algorithm of MTGODE

**Input:** Training data $X$, input length $T$, forecasting horizon $H$, batch size $B$, training epoch $E$, learning rate $\eta_i$, and the initialized MTGODE model $F(\cdot)$ with $\Theta_c, \Phi_c$, and $\Gamma_c$.

**Output:** Well-trained MTGODE model $F^*(\cdot)$.

```plaintext
1: data $\Leftarrow$ DataLoader($X, T, H, B$);
2: for $i \in 1, 2, \ldots, E$ do
3:     data.shuffle();
4:     for ($X, Y$) in enumerate(data.get_iterator()) do
5:         /* See Algorithm 2 for details */
6:         $\hat{Y} \Leftarrow F(X; \Theta_c, \Phi_c, \Gamma_c)$;
7:         $\ell \Leftarrow$ MAE($\hat{Y}, Y$);
8:         Calculate the stochastic gradients of $\Theta_c, \Phi_c$, and $\Gamma_c$ w.r.t. $\ell$;
9:         Update $\Theta_c, \Phi_c$, and $\Gamma_c$ w.r.t. their gradients and $\eta$;
10:    end for
11: end for
12: $\eta \Leftarrow$ LRScheduler($\eta, i$);
```

### 4.3 Comparison With Discrete Variants

Compared with the existing GNN-based methods [3], [8], [19], our approach is free from the over-smoothing issue (e.g., the model performance drops when the depth of graph propagation increases). This allows the CGP process to capture stable long-range spatial dependencies by disentangling the ties between graph propagation depth and integration time. Specifically, our method possesses the following properties:

**Property 1.** For a specific integration time $T_{cgp}$, MTGODE ensures the convergence of learned spatial representations by letting $K = T_{cgp}/\Delta t_{cgp}$ and $K \rightarrow \infty$.

**Proof.** See Appendix A, available in the online supplemental material.

**Property 2.** In MTGODE, letting $K = T_{cgp}/\Delta t_{cgp} \rightarrow \infty$ makes the numerical errors of spatial modeling approaching zero with a fixed integration time $T_{cgp}$.

**Proof.** See Appendix B, available in the online supplemental material.

### 4.4 Overall Architecture and Model Training

#### Overall Architecture

We have the proposed MTGODE method defined below by unifying the proposed continuous spatial and temporal message-passing mechanisms. It is worth noting that instead of simply concatenating them end-to-end, we take each intermediate state of the exterior CGP process as the initial state of interior CGP, thereby allowing the model to characterize the underlying interleaved spatial-temporal dynamics of input series in a fully continuous manner to derive more expressive representations for downstream forecasting tasks. Given two black-box ODE solvers, i.e., $\text{ODESolve}^1(\cdot)$ and $\text{ODESolve}^2(\cdot)$, the learned
spatial-temporal representations of input series can be obtained by integrating \( dH(t)/dt \)

\[
H_{out} = \text{ODESolve}^1 \left( H(0), \frac{dH(t)}{dt}, T_{sta} \right),
\]

where we have \( dH(t)/dt \) defined below based on the aforementioned two propositions

\[
\frac{dH(t)}{dt} = \mathcal{P} \left( \mathcal{A} \left( \text{ODESolve}^2 \left( \text{TCN} \left( H(t), t, \Theta \right), \frac{dH(t)}{dt}(x), 0, \ldots, T_{sta} \right), \Phi \right), R \right).
\]

In the above equations, the interior ODE solving and the attentive transformation, i.e., \( \mathcal{A}(., \Phi) \), are given by Equation (7) by letting \( H^G(0) = \text{TCN}(H(t), t, \Theta) \). In particular, we let the initial state \( H(0) = H^G(0) \) in Equation (11), and further define two black-box ODE solvers as the Euler or Runge-Kutta method adopted in [18] with different selected integration time and step sizes for simplicity.

Model Training. Given a sequence of historical observations \( X_{t+1:t+T} \), we first learn its representation \( H_{out} \in \mathbb{R}^{H \times D'} \) via Equation (14) and then make the forecasting with a downstream decoder \( g(., \Gamma_{dc}) \), as the end convolution shown in Fig. 2. Thus, our training objective described in Equation (3) can be reformulated as follows:

\[
f^*, g^* = \arg \min_{f,g} \sum_t \ell(g(f(X_{t+1:t+T}; \Theta, \Phi, \Gamma^t), \Gamma_{dc}), Y),
\]

where \( \Gamma^t = \{ \Gamma_{sc}, \Gamma_{gc} \} \), and \( \ell(.) \) denotes the mean absolute error (MAE). We illustrate the optimization of \( \text{MTGODE} \) in Algorithms 1 and 2.

Complexity Analysis. We analyze the time complexity of the proposed method. For the dynamic graph structure learning module, the time complexity is \( O(Nd^2 + N^2 d) \), where \( N \) and \( d \) are the number of variables and the dimension of embedding matrices in Equation (8). For a single-step graph propagation, the time complexity is \( O(ED' + ND'^2) \), where \( E \) and \( D' \) are the number of edges and latent state dimensions. Thus, the time complexity of the proposed CGP module is \( O(T_{sta}/\Delta t_{sta} \cdot (ED' + ND'^2)) \). For a single-step temporal aggregation, the time complexity is \( O(NQ_1D'^2/r) \). We use \( Q_1 \) and \( r \) to denote the length of latent states at \( l \)th layer in the temporal module and the dilatation factor of \( \text{TCN}(., \Theta) \). According to the padding trick mentioned in Equation (10), the time complexity of the proposed CTA module is \( O(T_{sta}/\Delta t_{sta} \cdot (NRD'^2/r)) \), where \( R \) denotes the temporal reception field of \( \text{MTGODE} \) discussed in Section 4.2. Compared with the discrete counterparts of our method, e.g., MTGNN [19], our model is less complex by eliminating redundant neural connections, such as the residual and skip layers with the time complexity of \( O(LNRD'^2) \), where \( L \) denotes the number of layers in discrete models. In the following section, we further evaluate the model efficiency of \( \text{MTGODE} \) empirically from various perspectives to demonstrate its superiority.

5 EXPERIMENTAL STUDY

In this section, we conduct comprehensive experiments on five real-world benchmark datasets to show the performance of \( \text{MTGODE} \). We compare our method with the state-of-the-art time series forecasting methods and follow their configurations to conduct experiments for fair comparisons. In this section, we also empirically demonstrate the potency and efficiency of two proposed continuous regimes, showing superior properties compared with discrete variants. Also, ablation and parameter sensitivity studies are conducted to further investigate the properties of \( \text{MTGODE} \).

5.1 Dataset Description

We experiment on five benchmark datasets to evaluate the performance of \( \text{MTGODE} \) and its competitors. Three of these are conventional time series datasets [12], i.e., Electricity, Solar-Energy, and Traffic, without predefined graph structures, and the rest two are traffic datasets [16], i.e., Metr-La and Pems-Bay, with predefined sensor maps (i.e., graph structures). We summarize the dataset statistics in Table 2 and provide a detailed description of them as follows:

- **Electricity**<sup>2</sup>: This dataset consists of the energy consumption records of 321 clients between 2012 and 2014 with the sampling rate set to 1 hour.
- **Solar-Energy**<sup>2</sup>: It contains the solar power production records of 137 PV plants in Alabama State in the year 2006, where the sampling rate is 10 minutes.
- **Traffic**<sup>2</sup>: A collection of hourly road occupancy rates measured by 862 sensors in the San Francisco Bay area between 2015 and 2016.
- **Metr-La**<sup>3</sup>: It contains the traffic speed readings with 5 minutes sampling rate from the 207 loop detectors in Los Angeles County highways in the year of 2012.
- **Pems-Bay**<sup>3</sup>: This dataset is provided by California Transportation Agencies Performance Measurement Systems, which consists of the traffic speed readings of 325 sensors in the Bay Area in the year 2017, where the data sampling rate is same as in Metr-La.

### Table 2: The Statistics of Five Multivariate Time Series Datasets

| Dataset     | # Samples | # Nodes | Sampling Rate | Predefined Graph |
|-------------|-----------|---------|---------------|------------------|
| Electricity | 26,304    | 321     | 1 hrs         | No               |
| Solar-Energy| 52,560    | 137     | 10 mins       | No               |
| Energy      | 17,544    | 862     | 1 hrs         | No               |
| Traffic     | 34,272    | 207     | 5 mins        | Yes              |
| Pems-Bay    | 52,116    | 4,732   | 5 mins        | Yes              |

5.2 Experimental Setup

In this subsection, we illustrate the detailed experimental setups, including baseline methods, evaluation protocols, and hyperparameter settings for replications.

2. https://github.com/laiguokun/multivariate-time-series-data
3. https://github.com/liyaguang/DCRNN

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
### TABLE 3
**Single-Step Forecasting Results on Three Benchmark Time Series Datasets**

| Dataset              | Metric | VARMLP | GRU | LSTM | TPA-LSTM | MTGNN | HyDCNN | STG-NCDE | MTGODE |
|----------------------|--------|--------|-----|------|----------|-------|--------|----------|--------|
| Electricity Horizon  | 3      | 0.1393 | 0.1102 | 0.0864 | 0.0823 | 0.0745 | 0.0832 | 0.6152 | 0.0736 |
|                      | CORR   | 0.8708 | 0.8597 | 0.9283 | 0.9439 | 0.9474 | 0.9354 | 0.8739 | 0.9430 |
|                      | 6      | 0.1620 | 0.1144 | 0.0931 | 0.0916 | 0.0878 | 0.0898 | 0.6584 | 0.0809 |
|                      | CORR   | 0.8389 | 0.8623 | 0.9135 | 0.9337 | 0.9316 | 0.9329 | 0.8663 | 0.9340 |
|                      | 12     | 0.1557 | 0.1183 | 0.1007 | 0.0964 | 0.0916 | 0.0921 | 0.7302 | 0.0891 |
|                      | CORR   | 0.8192 | 0.8472 | 0.9077 | 0.9250 | 0.9278 | 0.9285 | 0.8728 | 0.9279 |
| Traffic Horizon      | 3      | 0.1922 | 0.1932 | 0.1843 | 0.1803 | 0.1778 | 0.1806 | 0.2346 | 0.1620 |
|                      | CORR   | 0.9825 | 0.9823 | 0.9843 | 0.9850 | 0.9885 | 0.9865 | 0.9748 | 0.9868 |
|                      | 6      | 0.2679 | 0.2628 | 0.2559 | 0.2247 | 0.2348 | 0.2335 | 0.2908 | 0.2171 |
|                      | CORR   | 0.9655 | 0.9675 | 0.9690 | 0.9742 | 0.9726 | 0.9747 | 0.9805 | 0.9771 |
|                      | 12     | 0.4244 | 0.4163 | 0.3254 | 0.3234 | 0.3109 | 0.3094 | 0.5149 | 0.2901 |
|                      | CORR   | 0.9058 | 0.9150 | 0.9467 | 0.9487 | 0.9509 | 0.9515 | 0.8639 | 0.9577 |
| Solar-Energy Horizon | 3      |        |       |       |          |       |        |          |        |
|                      |        |        |       |       |          |       |        |          |        |
|                      |        |        |       |       |          |       |        |          |        |

**Bold** denotes the best performances. OOM indicates out-of-memory.

**Baselines.** We evaluate and compare **MTGODE** with representative and state-of-the-art time series baselines, such as **LSTNet** [12] and **HyDCNN** [14], on three time series datasets for **single-step forecasting**. We further compare with strong **GNN-based forecasting baselines** on two traffic datasets for **multi-step forecasting**, e.g., **MRA-BCGN** [20], **GMAN** [17], and **MTGNN** [19]. Note that our method and MTGNN do not rely on predefined graph structures so they are applicable and compared in both single/multi-step forecasting settings. We briefly introduce primary baselines as follows:

- **LSTNet** [12]: It combines convolution and recurrent neural networks to capture the short-term and long-term multivariate temporal dependencies.
- **TPA-LSTM** [13]: An attention-based recurrent neural network for multivariate time series forecasting.
- **HyDCNN** [14]: It forecasts time series with position-aware dilated temporal convolutions.
- **DCRNN** [12]: A graph diffusion-based gated recurrent neural network [35] for traffic forecasting.
- **STGCN** [3]: It stacks graph and temporal convolutions to capture spatial and temporal patterns jointly.
- **Graph WaveNet** [8]: It is similar to STGCN but consists of graph and dilated temporal convolutions.
- **GMAN** [17]: A spatial-temporal graph neural network equipped with spatial and temporal attention.
- **MRA-BCGN** [20]: A multi-range attentive bicomponent graph neural network for traffic forecasting.
- **MTGNN** [19]: A forecasting model based on graph neural networks and dilated temporal convolutions.
- **STGODE** [21]: An ODE-based spatial-temporal graph neural network for traffic forecasting.
- **STG-NCDE** [23]: A spatial-temporal graph neural network based on NCDEs for traffic forecasting.

**Configuration.** For **multi-step forecasting**, we adopt **Mean Absolute Error (MAE)**, **Root Mean Square Error (RMSE)**, and **Mean Absolute Percentage Error (MAPE)** as our evaluation metrics [19]. For **single-step forecasting**, we follow [12] and use **Root Relative Squared Error (RSE)** and **Empirical Correlation Coefficient (CORR)**, where better performance is indicated by higher CORR and lower RSE values. All experiments are independently repeated ten times on Linux servers with two AMD EPYC 7742 CPUs and eight NVIDIA A100 GPUs. Averaged performances are reported.

- **Single-Step Forecasting.** We choose the input length 168 and split all three benchmark time series datasets into training set (60%), validation set (20%), and testing set (20%) chronologically. The model is trained with Adam optimizer, batch size 4, and dropout rate 0.3. The hidden dimensions are fixed to 64. For Electricity and Traffic, our models are trained over 60 epochs with the base learning rate $10^{-3}$ and learning decays. Also, we let $\Delta t_{\text{step}} = 0.2$ and $\Delta t_{\text{step}} = 0.5$. For Solar-Energy, we run 40 epochs with a fixed learning rate $10^{-4}$, and we have $\Delta t_{\text{step}} = 0.167$ and $\Delta t_{\text{step}} = 0.25$. For graph learners, we adopt the settings suggested by [19].

- **Multi-Step Forecasting.** We set the input and output lengths to 12 with the data split 70%-10%-20%. On both datasets, we train 200 epochs using the Adam optimizer with a base learning rate of 0.001 and a dropout rate of 0.3. For Metr-La, the encoder and decoder hidden dimensions are 64 and 128. We use Euler solvers with integration time and step size set to 1.0 and 0.25. For Pems-Bay, the hidden dimensions are 128. On this dataset, we use Runge-Kutta solvers with integration time and step size set to 1.0 for simplicity. All experiments are with batch size 64 and learning rate decay. The configuration of graph learners is the same as in single-step forecasting.

### 5.3 Overall Comparisons
We first report the results of different methods on different horizons for single-step forecasting in Table 3. Specifically, we have two important observations:
In general, MTGODE achieves the best performance on three time series datasets, even when compared with HyDCNN and STG-NCDE, indicating its effectiveness in multivariate time series forecasting.

Our method significantly surpasses MTGNN in most cases with the same graph constructor, especially for long-term forecasting (i.e., horizons 6 and 12), demonstrating the superiority of our continuous regimes in capturing long-range and fine-grained spatial and temporal dependencies.

To further demonstrate the advantage of MTGODE, we compare it with competitive GNN-based methods on two benchmark traffic datasets under the setting of multi-step forecasting, where all baselines use predefined graph structures only except for MTGNN, STG-NCDE, and our method. We summarize the results in Table 4, from which we have the following observations:

- Similar to single-step forecasting, our method consistently outperforms MTGNN and even STG-NCDE under this setting with a similar graph learning schema, which further confirms the effectiveness of MTGODE in modeling multivariate time series data.

- Our method demonstrates better performance compared with STGODE. This can be attributed to two reasons: First, the proposed temporal ODE enables our approach to capture fine-grained temporal dynamics continuously. Second, our graph module is more expressive with the attentive transformation and free from graph priors; thus more robust to dataset biases.

- MTGODE surpasses DCRNN, STGCN, and Graph WaveNet significantly without relying on graph priors. Compared to MRA-BCGN and GMAN, our method achieves the best or on-par performance, demonstrating its competitiveness.

We also provide visualizations on two benchmark datasets in Fig. 3, where the forecasting results of different methods on a randomly selected node in each dataset are compared. In both cases, our method captures the general trend well and matches the ground truth better.

### 5.4 Effectiveness of Two Continuous Regimes

To empirically validate the discussion in Section 4.3 and study the behavioral differences between our method and its discrete variant, we dissect MTGODE by comparing the model performance and the number of parameters with different graph propagation and temporal aggregation depths. First, the left chart in Fig. 4 compares the proposed continuous graph propagation with its discrete implementation (Equation (4)). In particular, we disable the attentive transformation in this experiment to expose the essence of our proposed spatial ODE in Equation (6). Compared with MTGODE discrete (solid red curve), our method (solid blue line) is more robust (in terms of RSE) to the over-smoothing problem with increased propagation depths, where the gradually flattened performance curve and shrunk standard deviations indicate that our method allows the learned spatial representations to converge to a sweet spot by exploiting the long-range spatial dependencies, bringing significantly lower numerical errors (in terms of RSE) and better stability (w.r.t. standard deviations). At the same time, our method captures the general trend well and matches the ground truth better.

**Table 4**

| Dataset | Metric | DCRNN | STGCN | Graph WaveNet | GMAN | MRA-BCGN | MTGNN | STGODE | STG-NCDE | MTGODE |
|---------|--------|-------|-------|---------------|------|----------|-------|--------|----------|--------|
| **Metr-La** | | | | | | | | | | |
| 15 min | MAE    | 2.77  | 2.88  | 2.69          | 2.77 | 2.67     | 2.69  | 3.47   | 3.77     | 2.66   |
|        | RMSE   | 5.38  | 5.74  | 5.15          | 5.48 | 5.12     | 5.18  | 6.76   | 9.47     | 5.10   |
|        | MAPE (%) | 7.30  | 7.62  | 6.90          | 7.25 | 6.80     | 6.90  | 8.76   | 8.54     | 6.87   |
| 30 min | MAE    | 3.15  | 3.47  | 3.07          | 3.07 | 3.06     | 3.05  | 4.36   | 4.84     | 3.00   |
|        | RMSE   | 6.45  | 7.24  | 6.22          | 6.34 | 6.17     | 6.18  | 8.47   | 12.04    | 6.05   |
|        | MAPE (%) | 8.80  | 9.57  | 8.37          | 8.35 | 8.30     | 8.21  | 11.14  | 10.63    | 8.19   |
| 60 min | MAE    | 3.60  | 4.59  | 3.53          | 3.40 | 3.49     | 3.50  | 5.50   | 6.35     | 3.39   |
|        | RMSE   | 7.60  | 9.40  | 7.37          | 7.21 | 7.30     | 7.25  | 10.33  | 14.94    | 7.05   |
|        | MAPE (%) | 10.5  | 12.7  | 10.01         | 9.72 | 10.00    | 9.90  | 14.32  | 13.49    | 9.80   |
| **Pems-Bay** | | | | | | | | | | |
| 15 min | MAE    | 1.38  | 1.36  | 1.30          | 1.34 | 1.29     | 1.34  | 1.43   | 1.38     | 1.29   |
|        | RMSE   | 2.95  | 2.96  | 2.74          | 2.82 | 2.72     | 2.81  | 2.88   | 2.93     | 2.73   |
|        | MAPE (%) | 2.90  | 2.90  | 2.73          | 2.81 | 2.90     | 2.82  | 2.99   | 2.91     | 2.72   |
| 30 min | MAE    | 1.74  | 1.81  | 1.63          | 1.62 | 1.61     | 1.66  | 1.84   | 1.71     | 1.61   |
|        | RMSE   | 3.97  | 4.27  | 3.70          | 3.72 | 3.67     | 3.74  | 3.90   | 3.84     | 3.66   |
|        | MAPE (%) | 3.90  | 4.17  | 3.67          | 3.63 | 3.80     | 3.72  | 3.84   | 3.91     | 3.61   |
| 60 min | MAE    | 2.07  | 2.49  | 1.95          | 1.86 | 1.91     | 1.94  | 2.30   | 2.03     | 1.88   |
|        | RMSE   | 4.74  | 5.69  | 4.52          | 4.32 | 4.46     | 4.48  | 4.89   | 4.58     | 4.31   |
|        | MAPE (%) | 4.90  | 5.79  | 4.63          | 4.31 | 4.60     | 4.58  | 4.61   | 4.82     | 4.39   |

**Bold** denotes the best results.
time, our method also demonstrates a better parameter efficiency (in terms of # of parameters) compared with MTGODE discrete. It is worth noting that in this experiment, MTGODE* and MTGODE discrete have constant parameters as feature propagation is parameterless.

The middle chart in Fig. 4 compares our method with its discrete variant by varying the temporal aggregation depth. We can observe that with the increase in aggregation depth, MTGODE converges with decreased numerical errors (in terms of RSE). We can also observe that for MTGODE discrete, an increase in aggregation depth requires an increase in model parameters; hence it is complex. In contrast, MTGODE breaks this tie and thus allows a deeper aggregation to model more stable (w.r.t. standard deviations) and accurate temporal dynamics to capture fine-grained temporal patterns in a continuous and more parameter-efficient manner. In Section 5.7, we further demonstrate that MTGODE is more computational and memory efficient than its discrete counterparts.

5.5 Ablation Study
We construct four variants of our method to study the effectiveness of core components. Specifically, MTGODE w/o GSL disables the dynamic graph construction mechanism, MTGODE w/o CTA and MTGODE w/o CGP replace the temporal and spatial ODEs with their discrete implementations to study the potency of two continuous regimes. MTGODE w/o CGP & Attn further removes the attentive transformation in MTGODE w/o CGP to investigate the effectiveness of graph attentive transformation. In particular, the MTGODE discrete in Fig. 4 is equivalent to MTGODE w/o CTA & CGP & Attn, which has been investigated before so that we omit this variant in ablation study. The experimental results are in Table 5, where our method equipped with all components has the best performance across all datasets. In particular, our spatial ODE with the attentive transformation benefits the model best for learning effective representations. Besides, the performance gains obtained by the CGP itself and the embedded dynamic graph construction mechanism are also notable. A similar observation can also be made for CTA, where replacing it with its discrete version degrades the performance sharply.

5.6 Parameters Sensitivity
Apart from the experiments on graph propagation and temporal aggregation depths (i.e., the step sizes when solving MTGODE since the integration time is fixed) in Fig. 4, we also conduct experiments on other important hyperparameters in MTGODE, including temporal integration time $T_{\text{int}}$, spatial integration time $T_{\text{spa}}$, spatiotemporal encoder hidden dimension $D_e$, and batch size $B$, to investigate their impacts on our model, as shown in Fig. 5. Specifically, we have the following observations: (1) Moderately increasing the hidden state dimensions helps the model learning. We conjecture that this helps avoid the ODE trajectories intersecting with each other [36], thus encouraging our model to learn smoother ODE functions that can be easily solved; (2) For a specific spatial or temporal propagation depth, we can find a sweet spot when selecting the spatial or temporal integration time. It may be because a short wall time hinders the convergence of the learned representations and a long time introduces relatively large numerical errors; (3) Within a reasonable range, e.g., from 32 to 128, moderately increasing the batch size improves the model performance. We hypothesize that a relatively large batch size in our method helps reduce the variances of mini-batch gradients, which reduces the impact of noise on the model training.

5.7 Memory and Computational Efficiency
In Fig. 6, we compare the required multiply-accumulate operations (MACs) and GPU memory of MTGODE, its discrete variant, and MTGN[19]. In general, our method constantly has lower computational and memory overheads than MTGODE discrete and MTGN, especially for larger model depths, demonstrating the computational and memory efficiency of MTGODE. In comparison, discrete methods,

| Method | Electricity | Traffic | Solar-Energy |
|--------|-------------|---------|--------------|
| MTGODE | 0.0727 | 0.9436 | 0.4088 |
| w/o GSL | 0.0747 | 0.9414 | 0.4095 |
| w/o CTA | 0.0777 | 0.9428 | 0.4126 |
| w/o CGP | 0.0732 | 0.9427 | 0.4141 |
| w/o CGP & Attn | 0.0790 | 0.8979 | 0.4583 |

We replace the CTA and CGP in MTGODE with their discrete implementations, denoted as w/o CTA and w/o CGP. For w/o GSL, the learned graph structure at each training step is replaced by a randomly generated adjacency matrix. We further remove the attentive and continuous regime in CGP to construct the w/o CGP & Attn. A lower RSE and a higher CORR are expected.
e.g., MTGNN and our discrete variant, have more complex and discrete neural architectures, which inevitably introduce more intermediate operations and trainable parameters, resulting in higher computational and memory costs.

In particular, we find that MTGNN is slightly more efficient than MTGODE \texttt{discrete} in terms of large model depth because the latter one adopts the padding version of Equation (9), which inevitably involves more parameters in the following $L - 1$ temporal convolution layers except for the first layer. Although MTGODE is also based on this padding trick, it is still more computationally and memory efficient than MTGNN. In the rightmost chart in Fig. 4, we further demonstrate that the proposed method is more efficient than discrete methods to provide better forecasting results. Specifically, in this experiment, we slightly increase the spatial and temporal step size in our method (i.e., 0.25 $\rightarrow$ 0.34 and 0.25 $\rightarrow$ 0.5 in CTA and CGP processes, respectively) to trade model precision for speed. MTGODE remains surpassing strong GNN-based baselines by a significant margin in terms of forecasting errors.

6 CONCLUSION

Given the shortcomings of prior arts in multivariate time series forecasting, we investigate using neural ordinary differential equations and dynamic graph structure learning to model the continuous latent spatial-temporal dynamics of arbitrary multivariate time series. By solving the intersecting continuous graph propagation and temporal aggregation processes, our method allows the model to learn more expressive representations efficiently without relying on graph priors, showing better potential in real-world applications. Apart from the empirical justifications, we also theoretically analyze the main properties of our method and further demonstrate that it is more effective and efficient than the existing discrete approaches.

REFERENCES

[1] B. Heidrich, M. Turowski, N. Ludwigs, R. Mikut, and V. Hagenmeyer, “Forecasting energy time series with profile neural networks,” in Proc. 11th ACM Int. Conf. Future Energy Syst., 2020, pp. 220–230.
[2] Y. Li et al., “Weather forecasting using ensemble of spatial-temporal attention network and multi-layer perceptron,” Asia-Pacific J. Atmospheric Sci., vol. 57, no. 3, pp. 533–546, 2021.
[3] B. Yu, H. Yin, and Z. Zhu, “Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting,” in Proc. 27th Int. Joint Conf. Artif. Intell., 2018, pp. 3634–3640.
[4] H. Luikopeh, “Vector autoregressive models,” in Handbook of Research Methods and Applications in Empirical Macroeconomics, Cheltenham, U.K.: Edward Elgar Publishing, 2013.
[5] G. P. Zhang, “Time series forecasting using a hybrid ARIMA and neural network model,” Neurocomputing, vol. 50, pp. 159–179, 2003.
[6] R. Frigola, “Bayesian time series learning with Gaussian processes,” Ph.D. dissertation, Dept. Eng., St Edmund’s College, Univ. Cambridge, Cambridge, U.K., 2015.
[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” Nature, vol. 323, no. 6088, pp. 533–536, 1986.
[8] Z. Wu, S. Pan, G. Long, J. Jiang, and C. Zhang, “Graph wavenet for deep spatial-temporal graph modeling,” in Proc. 25th Int. Joint Conf. Artif. Intell., 2019, pp. 1907–1913.
[9] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling,” in Proc. Neural Inf. Process. Syst. Workshop Deep Learn., 2014.
[10] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” Neural Comput., vol. 9, no. 8, pp. 1735–1780, 1997.
[11] M. Schuster and K. K. Paliwal, “Bidirectional recurrent neural networks,” IEEE Trans. Signal Process., vol. 45, no. 11, pp. 2673–2681, Nov. 1997.
[12] G. Lai, W.-C. Chang, Y. Yang, and H. Liu, “Modeling long-and short-term temporal patterns with deep neural networks,” in Proc. 41st Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval, 2018, pp. 95–104.
[13] S.-Y. Shih, F.-K. Sun, and H.-Y. Lee, “Temporal pattern attention for multivariate time series forecasting,” Mach. Learn., vol. 108, no. 8, pp. 1421–1441, 2019.
[14] Y. Li, K. Li, C. Chen, X. Zhou, Z. Zeng, and K. Li, “Modeling temporal patterns with dilated convolutions for time-series forecasting,” ACM Trans. Knowl. Discov. Data, vol. 16, no. 1, pp. 1–22, 2021.
[15] W. Jiang and J. Luo, “Graph neural network for traffic forecasting: A survey,” 2021, arXiv:2101.11174.
[16] Y. Li, R. Yu, C. Shahabi, and Y. Liu, “Diffusion convolutional recurrent neural network: Data-driven traffic forecasting,” in Proc. Int. Conf. Learn. Representations, 2018.
[17] C. Zheng, X. Fan, C. Wang, and J. Qi, “GMAN: A graph multi-attention network for traffic prediction,” in Proc. 34th AAAI Conf. Artif. Intell., 2020, pp. 1234–1241.
[18] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, “Neural ordinary differential equations,” in Proc. 32nd Int. Conf. Neural Inf. Process. Syst., 2018, pp. 6572–6583.
[19] Z. Wu, S. Pan, G. Long, J. Jiang, X. Chang, and C. Zhang, “Connecting the dots: Multivariate time series forecasting with graph neural networks,” in Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining, 2020, pp. 753–763.
W. Chen, L. Chen, X. Xie, W. Cao, Y. Gao, and X. Feng, “Multi-range attentive bicomponent graph convolutional network for traffic forecasting,” in Proc. AAAI Conf. Artif. Intell., 2020, pp. 3529–3536.

Z. Fang, Q. Long, G. Song, and K. Xie, “Spatial-temporal graph ODE networks for traffic flow forecasting,” Proc. 37th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining, 2021, pp. 364–373.

C. Shang, J. Chen, and J. Bi, “Discrete graph structure learning for forecasting multiple time series,” in Proc. Int. Conf. Learn. Representations, 2020.

J. Choi, H. Choi, J. Hwang, and N. Park, “Graph neural controlled differential equations for traffic forecasting,” in Proc. 36th AAAI Conf. Artif. Intell., 2022, pp. 6367–6374.

P. Kidger, J. Morrill, J. Foster, and T. Lyons, “Neural controlled differential equations for irregular time series,” in Proc. 34th Int. Conf. Neural Inf. Process. Syst., 2020, Art. no. 562.

G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, Time Series Analysis: Forecasting and Control. Hoboken, NJ, USA: Wiley, 2015.

J. G. De Gooijer and R. J. Hyndman, “25 years of time series forecasting,” Int. J. Forecasting, vol. 22, no. 3, pp. 443–473, 2006.

A. Vaswani et al., “Attention is all you need,” in Proc. 31st Int. Conf. Neural Inf. Process. Syst., 2017, pp. 6000–6010.

H. Zhou et al., “Informer: Beyond efficient transformer for long sequence time-series forecasting,” in Proc. 35th AAAI Conf. Artif. Intell., 2021, pp. 11 106–11115.

H. Zhang, B. Wu, X. Yuan, S. Pan, H. Tong, and J. Pei, “Simplifying graph neural networks: Aspects, methods and trends,” 2022, arXiv:2205.07424.

Y. Liu et al., “Graph self-supervised learning: A survey,” IEEE Trans. Knowl. Data Eng., early access, May 06, 2022, doi: 10.1109/TKDE.2022.3172903.

L.-P. Xhonneux, M. Qu, and J. Tang, “Continuous graph neural networks,” in Proc. 37th Int. Conf. Mach. Learn., 2020, Art. no. 966.

Y. Wang, Y. Wang, J. Yang, and Z. Lin, “Dissecting the diffusion process in linear graph convolutional networks,” in Proc. Int. Conf. Neural Inf. Process. Syst., 2021, pp. 5758–5768.

F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, “Semi-supervised classification with graph convolutional networks,” in Proc. 36th Int. Conf. Mach. Learn., 2019, pp. 6861–6871.

T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in Proc. Int. Conf. Learn. Representations, 2017.

K. Cho et al., “Learning phrase representations using RNN encoder-decoder for statistical machine translation,” in Proc. Conf. Empir. Methods Natural Lang. Process., 2014, pp. 1724–1734.

E. Dupont, A. Doucet, and Y. W. Teh, “Augmented neural ODEs,” in Proc. 33rd Int. Conf. Neural Inf. Process. Syst., 2019, Art. no. 282.

Yun-Fang Li received the PhD degree in computer science from the National University of Singapore, in 2006. He is a senior lecturer with the Department of Data Science and Artificial Intelligence, Faculty of Information Technology, Monash University, Australia. His research interests include knowledge graphs, natural language processing and knowledge representation and reasoning.

Sihieng Chen received the bachelor’s degree in electronics engineering from the Beijing Institute of Technology, China, in 2011, the two master’s degrees in electrical and computer engineering and machine learning, and the doctorate degree in electrical and computer engineering from Carnegie Mellon University, in 2016, respectively. He is currently an associate professor with Shanghai Jiao Tong University. Before that, he was a research scientist with Mitsubishi Electric Research Laboratories (MERL) and an autonomy engineer with Uber Advanced Technology Group, working on autonomous driving. Before joining Uber, he was a postdoctoral research associate with Carnegie Mellon University. He was the recipient of the 2018 IEEE Signal Processing Society Young Author Best Paper Award and ASME SHM/NDE 2020 Best Journal Paper Award Runner-Up. His coauthored paper received the Best Student Paper Award with IEEE GlobalSIP 2018. His research interests include graph signal processing, graph neural networks and collaborative intelligence.

Bin Yang received the PhD degree in computer science from Fudan University, China. He is a professor with East China Normal University, China. He has been a professor with Aalborg University, Denmark and was previously with Aarhus University, Denmark and with Max Planck Institute for Informatics, Germany. His research interests include machine learning and data management.

Shirui Pan received the PhD degree in computer science from the University of Technology Sydney (UTS), Ultimo, NSW, Australia. He is a professor with the School of Information and Communication Technology, Griffith University, Australia. Prior to this, he is a senior lecturer with Monash University. His research interests include data mining and machine learning. He has published extensively in top-tier journals and conferences, including IEEE Transactions on Pattern Analysis and Machine Intelligence, IEEE Transactions on Knowledge and Data Engineering, IEEE Transactions on Neural Networks and Learning Systems, ICML, NeurIPS, KDD, AAAI, IJCAI, WWW, and ICDM. He is recognised as one of the AI 2000 AAAI/IJCAI Most Influential Scholars in Australia (2021). He is an ARC future fellow.

Ming Jin received the BEng degree from the Hebei University of Technology, Tianjin, China, in 2017, and the MInfTech degree from the University of Melbourne, Melbourne, Australia, in 2019. He is currently working toward the PhD degree in computer science with Monash University, Melbourne, Australia. His research focuses on graph neural networks (GNNs), time series analysis, data mining, and machine learning.

Yu Zheng received the BS and MS degrees in computer science from Northwest A&F University, China, in 2008 and 2011, respectively. She is currently working toward the PhD degree in computer science with La Trobe University, Melbourne, Australia. Her research interests include image classification, data mining, and machine learning.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/csdl.