

TEAM: Topological Evolution-aware Framework for Traffic Forecasting—Extended Version

Duc Kieu^{1,2,*}, Tung Kieu^{3,*}, Peng Han⁴, Bin Yang^{5,#}, Christian S. Jensen³, and Bac Le^{1,2}

¹University of Science, Ho Chi Minh City, Vietnam ²Vietnam National University, Ho Chi Minh City, Vietnam

³Aalborg University, Denmark ⁴University of Electronic Science and Technology of China, China

⁵East China Normal University, China

^{1,2}{18127080,lhbac}@hcmus.edu.vn ³{tungkvt,csj}@cs.aau.dk ⁴penghan@uestc.edu.cn ⁵byang@dase.ecnu.edu.cn

ABSTRACT

Due to the global trend towards urbanization, people increasingly move to and live in cities that then continue to grow. Traffic forecasting plays an important role in the intelligent transportation systems of cities as well as in spatio-temporal data mining. State-of-the-art forecasting is achieved by deep-learning approaches due to their ability to contend with complex spatio-temporal dynamics. However, existing methods assume the input is fixed-topology road networks and static traffic time series. These assumptions fail to align with urbanization, where time series are collected continuously and road networks evolve over time. In such settings, deep-learning models require frequent re-initialization and re-training, imposing high computational costs. To enable much more efficient training without jeopardizing model accuracy, we propose the Topological Evolution-Aware Framework (TEAM) for traffic forecasting that incorporates convolution and attention. This combination of mechanisms enables better adaptation to newly collected time series, while being able to maintain learned knowledge from old time series. TEAM features a continual learning module based on the Wasserstein metric that acts as a buffer that can identify the most stable and the most changing network nodes. Then, only data related to stable nodes is employed for re-training when consolidating a model. Further, only data of new nodes and their adjacent nodes as well as data pertaining to changing nodes are used to re-train the model. Empirical studies with two real-world traffic datasets offer evidence that TEAM is capable of much lower re-training costs than existing methods are, without jeopardizing forecasting accuracy.

PVLDB Reference Format:

Duc Kieu^{1,2}, Tung Kieu³, Peng Han⁴, Bin Yang⁵, Christian S. Jensen³, and Bac Le^{1,2}. TEAM: Topological Evolution-aware Framework for Traffic Forecasting—Extended Version. PVLDB, 18(2): 1 - 16, 2024.
doi:10.14778/3705829.3705844

PVLDB Artifact Availability:

The source code, data, and/or other artifacts have been made available at <https://github.com/kvmduc/TEAM-topo-evo-traffic-forecasting>.

*: Equal contributions, #: Corresponding author

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit <https://creativecommons.org/licenses/by-nc-nd/4.0/> to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.

Proceedings of the VLDB Endowment, Vol. 18, No. 2 ISSN 2150-8097.
doi:10.14778/3705829.3705844

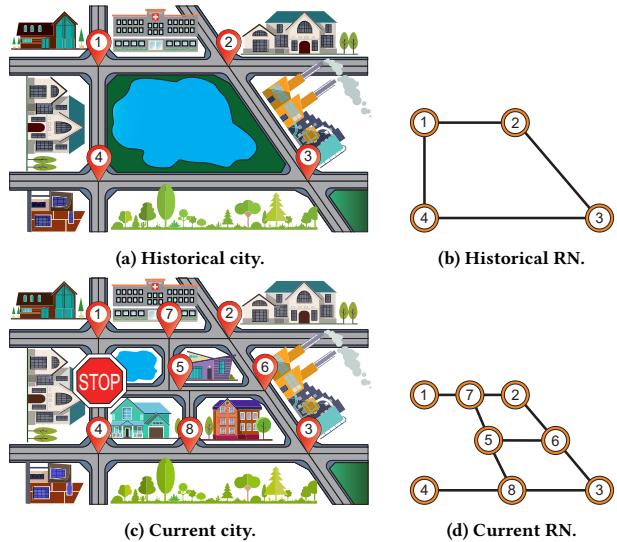


Figure 1: Example of a city and its corresponding evolving RN.

1 INTRODUCTION

Transportation has recently experienced substantial development, owing to technological advancements. One of the main developments is the widespread diffusion of sensor-equipped devices, resulting in the massive production of transportation data. This information further supports inexpensive and effective transportation management solutions [29]. For example, proximity sensors or speeding cameras are installed on different roads and intersections to continuously collect traffic information such as traffic flow and speed. The result is the availability of large amounts of time-ordered traffic observations, known as traffic time series.

Traffic forecasting fueled by traffic data, is a core element of many applications in intelligent transportation and plays an important role in spatio-temporal data mining. Forecasted information (e.g., traffic flow and traffic speed), which offers insight into the dynamic characteristics of the underlying traffic network, can contribute to, e.g., alarms for hotspots [31], route planning [19], and route recommendation [56]. To capture complex spatio-temporal dynamics, many deep learning based methods have recently been proposed and show promising results on challenge datasets due to their ability to learn non-linear dynamics [35, 59]. These methods are often built on Graph Neural Networks (GNNs) [16, 32] and Temporal Neural Networks (TCNs) [24, 50] to capture spatial and the temporal information, respectively. Although achieving competitive performance, these methods face two challenges.

First, traffic forecasting is a long-term task, where traffic behaviors can change over time due to changes in the graph structure of road networks (RNs). For example, RNs that represent cities often expand to support growing populations caused by urbanization. This results in new nodes and edges being added to existing RNs. Meanwhile, old roads and regions can become obsolete, leading to the elimination of nodes and edges from existing RNs. Also, in some countries, roads are frequently blocked, and directions are frequently changed depending on the day of the week. Observations such as these motivate solutions to the problem of traffic forecasting for RNs with evolving topology.

Fig. 1 shows an example of a city and its evolving RN. However, existing forecasting proposals only work on static RNs [35, 59, 60] and thus do not accommodate real-world scenarios. A trivial solution is to re-initialize a new model and re-train it on the new RN data whenever the RN has evolved. However, this imposes challenges to both storage and computation because of high data processing and model training overheads, particularly when RNs grow. Another way of accommodating evolving RNs is to train a model on historical regions and then transfer the learned knowledge to a new model, which is further trained on data from updated RN parts. However, the direct use of transferred models faces two limitations. (i) The historical and new temporal data of a node may not exhibit similar patterns because traffic data evolves and exhibits the data shift characteristics [5, 55]. Transferred models cannot learn inconsistent patterns between the historical and the new data, thus exhibiting substandard performance. Also, when the topology of an RN changes, traffic behaviors corresponding to the new topology deviate from the previous behaviors. Here, spatial dependencies captured by transferred models may no longer be appropriate. (ii) Useful information, such as stable patterns that are captured by the transferred models, may be forgotten after being transferred, rather than being consolidated [9]. Models that forget stable patterns may experience substandard performance.

Second, we observe that traffic forecasting models can achieve impressive accuracy due to the practice of fitting on massive datasets. However, in practical scenarios, the cost of constructing and maintaining a forecasting model is high. As mentioned above, a natural consideration is to transfer the knowledge and only require model training on updated regions. However, the data for the evolution of RNs, which is used to transfer the previous model, is substantially smaller than that used for a re-initialized model. An effective model is then required to incrementally capture dynamic and complex spatio-temporal correlations given a small-scale dataset. However, recent studies [21, 47, 54] focus on building forecasting models with high accuracy, overlooking the ability to learn effectively to model complex and non-linear spatio-temporal dependencies with small-scale data.

We propose a Topological Evolution-aware Framework (TEAM) for traffic forecasting to solve the above two problems. To tackle the first problem, we propose a continual learning module that adopts the rehearsal-based continual learning mechanism. This module works as a buffer and stores a limited number of historical data samples. The module is then integrated into the traffic forecasting framework, where it provides stored samples to enforce forecasting model rehearsal when new data is similar to historical data. By rehearsing, the historical knowledge is consolidated, and the model

can mitigate forgetting. In case the new data is different from the historical data (i.e., exhibits distribution shift [20]), the historical knowledge is no longer useful, and thus rehearsal is unnecessary.

To measure the difference between historical and new data, we transform the historical and new data into two histograms and use the Wasserstein metric to compute their similarity. A limited number of historical data samples of several historical graph nodes that have high similarity (i.e., the most stable nodes) are selected and stored in the buffer. In addition, those that have the lowest similarity (i.e., the most changing nodes) are selected for update with the new data. A new adjacency matrix is constructed from the most stable nodes, the most changing nodes, and the newly added nodes. Finally, data in the buffer and evolved data of the new adjacency matrix serve as the input for training the transferred model. The constructed adjacency matrix is significantly smaller than the adjacency matrix of the entire RN. Thus, the complexity of the model training is reduced substantially.

To overcome the second issue, we propose a model, called **C**onvolution **A**ttention for **S**patio-**T**emporal (CAST), that uses a hybrid architecture that combines convolution and attention modules for both spatial and temporal computations. Existing studies [15, 52] show that combining convolution and attention allows a model to learn faster and to converge more easily so that the model can work well on small-scale datasets. Hybridizing convolution and attention also enables a model to better model dynamic and non-linear spatial-temporal correlations. Further, convolution focuses on local patterns such as seasonalities, variations w.r.t. the temporal aspect, and closed-neighbor-nodes w.r.t. the spatial aspect [42]. In contrast, attention focuses on global patterns such as trends w.r.t. the temporal aspect and far-neighbor-nodes w.r.t. the spatial aspect [12]. By combining convolution and attention, we aim to obtain a model that exploits both local and global patterns to yield better accuracy.

To the best of our knowledge, this is the first study to enable traffic forecasting for evolving RNs. We make four contributions. First, we formulate the problem of traffic forecasting in evolving RNs. Second, we propose CAST, a framework for traffic forecasting, where the main model can learn effectively on small-scale data using a hybrid architecture. Third, we propose a continual learning module that enables the forecasting model to effectively learn on evolving RNs. Then, we integrate the module into CAST to form TEAM. Fourth, we report on extensive empirical studies that offer insight into pertinent design properties of the proposed framework and offer evidence that the proposed framework is able to outperform the state-of-the-art approaches w.r.t. accuracy and runtime.

The rest of the paper is organized as follows. Section 2 describes the preliminaries and formalizes the problem. Section 3 presents the methodology. Section 4 describes the experimental study and the results. Section 5 discusses related work, and Section 6 concludes the paper.

2 PRELIMINARIES

2.1 Traffic Forecasting

Consider an RN modeled as a graph $G = (V, E)$, where V is a finite set of $|V| = N$ vertices and E is a set of edges, and let G be

represented by an adjacency matrix A . Assume corresponding data for all vertices over P historical time steps $\mathcal{X} = \langle \mathbf{X}_{T-P+1}, \dots, \mathbf{X}_T \rangle$.

The data at time step T is denoted as $\mathbf{X}_T \in \mathbb{R}^{N \times F}$, where F is the number of traffic signal features. The goal of traffic forecasting is to build a model that works as a function $f_\theta(\cdot)$ that learns from P previous steps of data \mathcal{X} to forecast data for H future steps of data $\hat{\mathcal{X}} = \langle \hat{\mathbf{X}}_{T+1}, \dots, \hat{\mathbf{X}}_{T+H} \rangle$ according to Eq. 1.

$$\langle \mathbf{X}_{T-P+1}, \dots, \mathbf{X}_T \rangle \xrightarrow{f_\theta(\cdot)} \langle \hat{\mathbf{X}}_{T+1}, \dots, \hat{\mathbf{X}}_{T+H} \rangle \quad (1)$$

2.2 Graph Evolution

The evolution of graph over time is conceptually represented by a series of graphs $\langle G^1, G^2, \dots, G^Q \rangle$, so that $G^\pi = (V^\pi, E^\pi)$ represents the snapshot of the graph at period π . A period is a generic term. We can flexibly define the length of a period to a *day*, *a week*, *a month*, a *quarter*, a *year*, or any other length. Since $\langle G^1, G^2, \dots, G^Q \rangle$ represent snapshots of a specific graph, we have that $G^\pi = G^{\pi-1} + \Delta G^\pi$. Here, $\Delta G^\pi = (\Delta V^\pi, \Delta E^\pi)$ is the incremental data that captures the change (i.e., the difference) w.r.t. edges and nodes (either addition or removal) between the graph G^π and the previous graph $G^{\pi-1}$. More specifically, ΔG^π encompasses added and removed nodes, added and removed edges, and all existing nodes incident on the added and removed edges. Fig. 2 exemplifies $G^{\pi-1}$, G^π , and the evolved part ΔG^π .

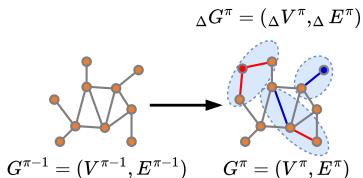


Figure 2: Evolution from $G^{\pi-1}$ to G^π . The blue oval regions highlight the evolved parts. Red nodes and edges denote added nodes and edges, respectively. Blue nodes and edges denote removed nodes and edges, respectively.

2.3 Problem Statement

Assume a long-term RN modeled as a sequence of graph snapshots $\langle G^1, G^2, \dots, G^Q \rangle$, where each $G^\pi = G^{\pi-1} + \Delta G^\pi$ represents the evolution of the RN from period $\pi - 1$ to period π due to many reasons such as the expansion or shrinkage of a region, newly-built roads, and discontinued roads.

For each ΔG^π , P corresponding historical time steps $\Delta \mathcal{X}^\pi = \langle \Delta \mathbf{X}_{T-P+1}^\pi, \dots, \Delta \mathbf{X}_T^\pi \rangle$ are given. We aim to build a framework that works as a function series $\langle f_\theta^1(\cdot), f_\theta^2(\cdot), \dots, f_\theta^Q(\cdot) \rangle$, where each $f_\theta^\pi(\cdot)$ is transferred from $f_\theta^{\pi-1}(\cdot)$ and learns from the data from the P previous steps of the evolved parts in $\Delta \mathcal{X}^\pi$ to forecast data for H steps into the future for the entire RN in $\hat{\mathcal{X}}^\pi = \langle \hat{\mathbf{X}}_{T+1}^\pi, \dots, \hat{\mathbf{X}}_{T+H}^\pi \rangle$ as shown in Eq. 2.

$$\begin{cases} \langle \mathbf{X}_{T-P+1}^\pi, \dots, \mathbf{X}_T^\pi \rangle \xrightarrow{f_\theta^\pi(\cdot)} \langle \hat{\mathbf{X}}_{T+1}^\pi, \dots, \hat{\mathbf{X}}_{T+H}^\pi \rangle, & \text{if } \pi = 1 \\ \langle \Delta \mathbf{X}_{t-P+1}^\pi, \dots, \Delta \mathbf{X}_T^\pi \rangle \xrightarrow{f_\theta^\pi(\cdot)} \langle \hat{\mathbf{X}}_{T+1}^\pi, \dots, \hat{\mathbf{X}}_{T+H}^\pi \rangle, & \text{if } \pi > 1 \end{cases} \quad (2)$$

3 METHODOLOGY

3.1 Framework Overview

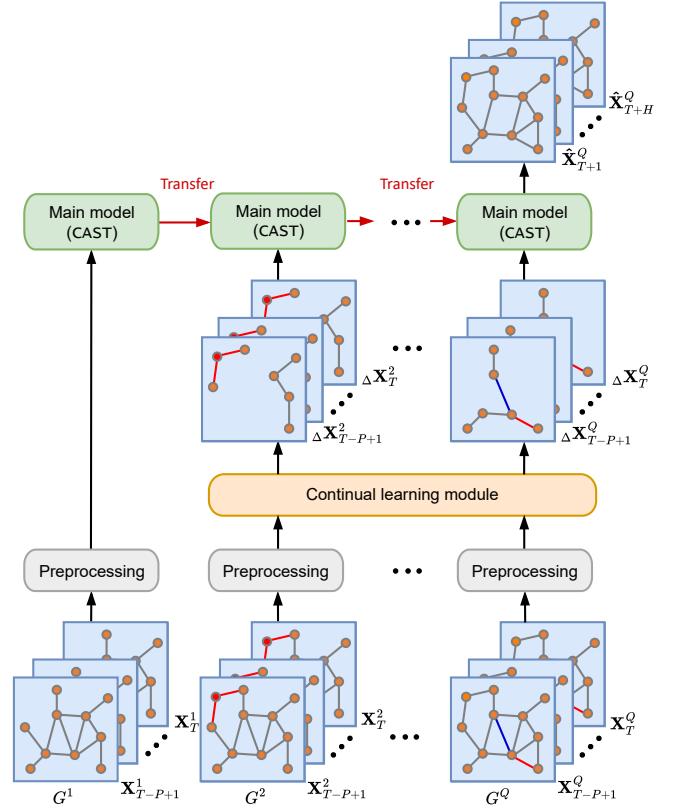


Figure 3: TEAM framework overview.

We show an overview of the proposed framework in Fig. 3. First, the data is fed into the *preprocessing* component that uses a common preprocessing technique to reduce the difference in terms of the magnitude between observations. An observation $\mathbf{X}_t^\pi \in \mathcal{X}^\pi$ at each G^π is re-scaled by using the mean μ^π and standard deviation σ^π

of \mathcal{X}^π as follows $\mathbf{X}_t^\pi = \frac{(\mathbf{X}_t^\pi - \mu^\pi)}{\sigma^\pi}$. Next, the preprocessed data

$\langle \mathbf{X}_{T-P+1}^1, \dots, \mathbf{X}_T^1 \rangle$ from the first period is fed into the *main model*, which is covered in Section 3.2. The main model is fully trained by the data from the first period. Next, the preprocessed data from the following periods are fed into the *continual learning module*, to be presented in Section 3.3. This module identifies the most stable and most changing nodes as well as their incidental edges and combines these with the evolved parts to produce the incremental data $\langle \Delta \mathbf{X}_{T-P+1}^\pi, \dots, \Delta \mathbf{X}_T^\pi \rangle$, $\pi > 1$ in the π -th period for partially training the main model. To do that, the trained model from the previous period is transferred and takes the incremental data as input. The incremental data is significantly smaller than the full data. Thus, training the model with incremental data is much faster. In other words, the cost of training the models in the following periods is substantially reduced. This procedure continues until the last period Q .

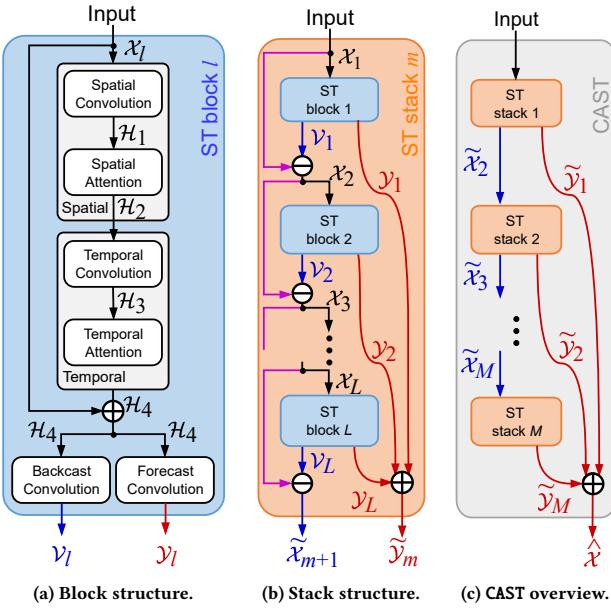


Figure 4: Main model (CAST).

3.2 Main Model

We propose a hybrid architecture [54] that combines attention and convolution [15], namely Convolution Attention for Spatio-Temporal traffic forecasting (CAST) as shown in Fig. 4. Intuitively, the convolution layers can be viewed as filters that produce high-level features to be fed into the attention module, which is treated as an implicit memory, capable of storing a representation of complex knowledge. By doing so, the model can swiftly adapt to new patterns using limited data as the RN evolves, while still preserving intricate historical knowledge. Further, convolution focuses on local patterns such as seasonalities, variations w.r.t. the temporal aspect, and close-neighbor-nodes w.r.t. the spatial aspect. In contrast, attention focuses on global patterns. By combining the two, we exploit both local and global information to improve forecasting accuracy. CAST consists of spatio-temporal stacks (Fig. 4c). Each such stack consists of spatio-temporal blocks (Fig. 4b). Each block is a basic component that performs spatial and temporal computation (Fig. 4a).

Spatio-temporal Block. A spatio-temporal (ST) block consists of three components, including a spatial component to learn spatial information, a temporal component to learn temporal information, and a component to compute forecasting and backcasting values. The spatial and temporal components combine convolution and attention layers with the constraint that the attention layers must follow the convolution layers. This ordering allows the model to converge faster (i.e., capture more high-level features and then memorize these by only training on a small amount of data) [15, 52]. Next, we describe the operation of the l -th block in detail. The first ST block is a special case and receives the graph signal $X_l^\pi \equiv X^\pi, \pi = 1$ or $X_l^\pi \equiv \Delta X^\pi, \pi > 1$ as input. The inputs

X_l to the remaining ST blocks are the residual outputs from the previous blocks. The input is first fed into the spatial component, which performs spatial convolution and attention. For the spatial convolution, we employ ChebnetII [23], which is an improved version of ChebNet [16] that has more expressive capabilities than

GCN [32]. More specifically, the computation is defined as follows. First, Laplacian matrices \mathbf{L}^π and $\Delta \mathbf{L}^\pi$ are computed as shown in Eq. 3.

$$\begin{cases} \mathbf{L}^\pi = \mathbf{I}^\pi - \mathbf{D}^{\pi^{-\frac{1}{2}}} \mathbf{A}^\pi \mathbf{D}^{\pi^{-\frac{1}{2}}}, & \text{if } \pi = 1 \\ \Delta \mathbf{L}^\pi = \Delta \mathbf{I}^\pi - \Delta \mathbf{D}^{\pi^{-\frac{1}{2}}} \Delta \mathbf{A}^\pi \Delta \mathbf{D}^{\pi^{-\frac{1}{2}}}, & \text{if } \pi > 1 \end{cases} \quad (3)$$

Here, $\mathbf{D}^\pi \in \mathbb{R}^{N^\pi \times N^\pi}$ and $\Delta \mathbf{D}^\pi \in \mathbb{R}^{\Delta N^\pi \times \Delta N^\pi}$ are degree matrices, $\mathbf{A}^\pi \in \mathbb{R}^{N^\pi \times N^\pi}$ and $\Delta \mathbf{A}^\pi \in \mathbb{R}^{\Delta N^\pi \times \Delta N^\pi}$ are adjacency matrices, and $\mathbf{I}^\pi \in \mathbb{R}^{N^\pi \times N^\pi}$ and $\Delta \mathbf{I}^\pi \in \mathbb{R}^{\Delta N^\pi \times \Delta N^\pi}$ are identity matrices. We follow existing studies to construct the adjacency matrices, i.e., generating \mathbf{A}^π and $\Delta \mathbf{A}^\pi$, using distances between entities [35]. We thus compute the RN distances to construct \mathbf{A}^π and $\Delta \mathbf{A}^\pi$ by using a Gaussian kernel. The weight of the edge between sensors i and j is defined as follows.

$$\begin{cases} \mathbf{A}_{ij}^\pi = \exp - \frac{dist(V_i, V_j)^2}{\sigma^2}, & \text{if } \pi = 1 \\ \Delta \mathbf{A}_{ij}^\pi = \exp - \frac{dist(\Delta V_i, \Delta V_j)^2}{\sigma^2}, & \text{if } \pi > 1 \end{cases} \quad (4)$$

Here, $dist(V_i, V_j)$ and $dist(\Delta V_i, \Delta V_j)$ are the distances between entities i and j , and σ is the standard deviation of the distances. We set $\mathbf{A}_{ij}^\pi = 0$ and $\Delta \mathbf{A}_{ij}^\pi = 0$ if they are below a threshold ϵ . Next, a rescaled Laplacian matrix $\hat{\mathbf{L}}^\pi \in \mathbb{R}^{N^\pi \times N^\pi}$ and $\Delta \hat{\mathbf{L}}^\pi \in \mathbb{R}^{\Delta N^\pi \times \Delta N^\pi}$ are computed as shown in Eq. 5.

$$\begin{cases} \hat{\mathbf{L}}^\pi = \frac{2\mathbf{L}^\pi}{\lambda_{max}^\pi} - \mathbf{I}, & \text{if } \pi = 1 \\ \Delta \hat{\mathbf{L}}^\pi = \frac{2\Delta \mathbf{L}^\pi}{\Delta \lambda_{max}^\pi} - \Delta \mathbf{I}, & \text{if } \pi > 1 \end{cases} \quad (5)$$

Here, λ_{max}^π and $\Delta \lambda_{max}^\pi$ are the maximum of eigenvalues of \mathbf{L}^π and $\Delta \mathbf{L}^\pi$, respectively. For a graph signal with F_{in} channels, $X_l^\pi \in \mathbb{R}^{N^\pi \times F_{in} \times P}$ or $\Delta X_l^\pi \in \mathbb{R}^{\Delta N^\pi \times F_{in} \times P}$, the output of the spatial convolution is computed as shown in Eq. 6.

$$\mathcal{H}_1 = \begin{cases} \frac{2}{O+1} \sum_{o=0}^O \sum_{q=0}^O \gamma_q P_o(\Gamma(q, O)) P_o(\hat{\mathbf{L}}) X_l^\pi, & \text{if } \pi = 1 \\ \frac{2}{O+1} \sum_{o=0}^O \sum_{q=0}^O \gamma_q P_o(\Gamma(q, O)) P_o(\hat{\mathbf{L}}) \Delta X_l^\pi, & \text{if } \pi > 1 \end{cases} \quad (6)$$

Here, $\gamma_q \in \mathbb{R}^{F_{in} \times F_1}$ is a learnable weight matrix, $\Gamma(q, O) = \cos(\frac{\pi(q + \frac{1}{2})}{O+1})$,

and $P_o(\cdot)$ is a Chebyshev function, which is defined as a recursive function as follows.

$$\begin{cases} P_0(x) = 2P_{o-1}(x) + P_{o-2}(x) \\ P_1(x) = x \\ P_0(x) = 1 \end{cases} \quad (7)$$

The output of the spatial convolution $\mathcal{H}_1 \in \mathbb{R}^{N^\pi \times F_1 \times P}, \pi = 1$ or $\in \mathbb{R}^{\Delta N^\pi \times F_1 \times P}, \pi > 1$ is then fed into the spatial attention. For simplicity, we denote $\mathcal{H}_1 \in \mathbb{R}^{N^\pi \times F_1 \times P}, \pi = 1$ or $\in \mathbb{R}^{\Delta N^\pi \times F_1 \times P}, \pi > 1$ using the unified notation $\mathcal{H}_1 \in \mathbb{R}^{(N^\pi || \Delta N^\pi) \times F_1 \times P}$ from now.

We employ Graph Attention Network (GAT) [51] for the spatial attention. First, two matrices $\mathbf{W}_{s_{\text{key}}}, \mathbf{W}_{s_{\text{query}}} \in \mathbb{R}^{(N^\pi || \Delta N^\pi) \times 1 \times P}$ are computed from $\mathbf{W}_h \in \mathbb{R}^{(N^\pi || \Delta N^\pi) \times F_2 \times P}$.

$$\mathbf{W}_h = \mathcal{H}_1 \mathbf{W}_s; \quad \mathbf{W}_{s_{\text{key}}} = \mathbf{W}_h \mathbf{a}_{s_{\text{key}}}; \quad \mathbf{W}_{s_{\text{query}}} = \mathbf{W}_h \mathbf{a}_{s_{\text{query}}} \quad (8)$$

Here, $\mathbf{W}_s \in \mathbb{R}^{F_1 \times F_2}$, $\mathbf{a}_{s_{\text{key}}} \in \mathbb{R}^{F_2 \times 1}$, and $\mathbf{a}_{s_{\text{query}}} \in \mathbb{R}^{F_2 \times 1}$ are learnable parameters.

Then, a spatial attention matrix $\mathbf{E}_s \in \mathbb{R}^{(N^\pi || \Delta N^\pi) \times (N^\pi || \Delta N^\pi)}$ is constructed as shown in Eq. 9. The spatial attention matrix plays the role of specifying the importance between all the nodes in the RN.

$$\mathbf{E}_s = \mathbf{W}_{s_{\text{key}}} \mathbf{W}_{s_{\text{query}}}^\top \quad (9)$$

A normalized spatial attention matrix $\mathbf{E}'_s \in \mathbb{R}^{(N^\pi || \Delta N^\pi) \times (N^\pi || \Delta N^\pi)}$ is computed from each element $\mathbf{E}_{s_{i,j}}$.

$$\mathbf{E}'_s = \frac{\exp(\mathbf{E}_{s_{i,j}})}{\sum_j^N \exp(\mathbf{E}_{s_{i,j}})} \quad (10)$$

The output of spatial attention $\mathcal{H}_2 \in \mathbb{R}^{(N^\pi || \Delta N^\pi) \times F_2 \times P}$ is computed as follows.

$$\mathcal{H}_2 = \text{ReLU}(\mathbf{E}'_s \mathbf{W}_h) \quad (11)$$

Here, ReLU is the rectified linear unit activation function [40]. The spatial attention shown in Eq. 11 can be stabilized by using the multi-head mechanism [51], where multiple spatial attentions work together in parallel.

$$\mathcal{H}_2 = \left\|_{u=1}^U \text{ReLU}(\mathbf{E}'_s^u \mathbf{W}_h^u) \right\| \quad (12)$$

Here, $\left\| \cdot \right\|$ denotes the concatenation operator, U is the number of spatial attention heads, $\mathbf{E}'_s^u \in \mathbb{R}^{(N^\pi || \Delta N^\pi) \times (N^\pi || \Delta N^\pi)}$ is the u -th normalized spatial attention matrix, and $\mathbf{W}_h^u \in \mathbb{R}^{(N^\pi || \Delta N^\pi) \times \frac{F_2}{U} \times P}$, where $\mathbf{W}_h^u = \mathcal{H}_1 \mathbf{W}^u$ is the weight matrix of the u -th spatial attention heads.

The output of the spatial attention is also the output of the spatial component, which is fed into the temporal component. In the temporal component, the input is first fed into a TCN with l layers. We use dilated causal convolution for TCN. Dilated causal convolution works by convoluting V attribute vectors from B different timestamps, where the attribute vectors to be convoluted are d timestamps apart. At different layers, B is often the same, while d may be different.

For example, Figure 5 shows a TCN with $B = 2$, meaning that the dilated causal convolution always convolutes two attribute vectors from two timestamps. The dilation factors d in the 1st, 2nd, and 3rd layers are 1, 2, and 4, respectively. Thus, in the 1st layer, the two attribute vectors to be convoluted are 1 timestamp apart; in the 2nd layer, 2 timestamps apart; in the 3rd layer, the 4 timestamps apart.

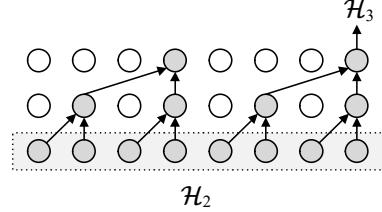


Figure 5: Temporal convolution.

The temporal convolution is defined as follows.

$$\mathcal{H}_3 = \mathcal{H}_2 \star_t \mathbf{W}_{t_1} = \sum_{F_3=1}^{F_3} \sum_{b=1}^B \mathbf{W}[b, F_3] \odot \mathcal{H}_{2_{t-d \times (b-1)}}[F_3], \quad (13)$$

where, \star_t represents a convolutional operator at timestamp t , \odot is Hadamard product (i.e., element-wise multiplication), and $\mathbf{W}_{t_1} \in \mathbb{R}^{B \times F_3}$ is the convolution filter. We compute Eq. 13 with a number of filters in the filter bank \mathcal{W}_{t_1} .

The output of temporal convolution, $\mathcal{H}_3 \in \mathbb{R}^{(N^\pi || \Delta N^\pi) \times F_3 \times P}$, is fed into temporal attention. The temporal attention can be viewed as the importance score between all timestamps. First, the temporal attention matrix $\mathbf{E}_t \in \mathbb{R}^{P \times P}$ is computed as follows.

$$\mathbf{E}_t = \mathbf{W}_{t_2} \sigma((\mathcal{H}_3^\top \mathbf{W}_{t_{\text{key}}}) \mathbf{W}_{t_3} (\mathbf{W}_{t_{\text{query}}} \mathcal{H}_3) + \mathbf{b}_t) \quad (14)$$

Here, $\mathbf{W}_{t_2}, \mathbf{b}_t \in \mathbb{R}^{P \times P}$, $\mathbf{W}_{t_{\text{key}}} \in \mathbb{R}^{F_3 \times F_4}$, $\mathbf{W}_{t_3} \in \mathbb{R}^{F_4}$, and $\mathbf{W}_{t_{\text{query}}} \in \mathbb{R}^{F_3}$ are learnable weight matrices, and \mathbf{b}_t is the bias. A normalized temporal attention matrix $\mathbf{E}'_t \in \mathbb{R}^{P \times P}$ is computed from each element $\mathbf{E}_{t_{i,j}}$.

$$\mathbf{E}'_t = \frac{\exp(\mathbf{E}_{t_{i,j}})}{\sum_j^P \exp(\mathbf{E}_{t_{i,j}})} \quad (15)$$

Then, the output of temporal attention $\mathcal{H}_4 \in \mathbb{R}^{(N^\pi || \Delta N^\pi) \times F_4 \times P}$ is computed as follows.

$$\mathcal{H}_4 = \mathcal{H}_3 \mathbf{E}'_t \quad (16)$$

The output of temporal attention \mathcal{H}_4 is then conducted as a residual connection as shown in Eq. 17.

$$\mathcal{H}_4 = \mathcal{H}_4 + \mathcal{X}_l^\pi \quad (17)$$

Then the output of the residual connection is synchronously fed into the forecast and the backcast convolution to produce $\mathcal{Y}_l \in \mathbb{R}^{(N^\pi || \Delta N^\pi) \times p \times H}$ and $\mathcal{V}_l \in \mathbb{R}^{(N^\pi || \Delta N^\pi) \times F_{in} \times P}$, which are the forecast and the backcast of block l , respectively.

$$\mathcal{Y}_l = \mathcal{H}_4 \star \mathbf{W}_{\text{forecast}} + \mathbf{b}_{\text{forecast}} \quad (18)$$

$$\mathcal{V}_l = \mathcal{H}_4 \star \mathbf{W}_{\text{backcast}} + \mathbf{b}_{\text{backcast}} \quad (19)$$

Here, \star indicates 2D convolution operator; $\mathbf{W}_{\text{forecast}} \in \mathbb{R}^{F_4 \times p}$ and $\mathbf{b}_{\text{forecast}} \in \mathbb{R}^p$ are the learnable weight matrix and bias vector of the forecast convolution component, where p is the number of output feature; and $\mathbf{W}_{\text{backcast}} \in \mathbb{R}^{F_4 \times F_{in}}$ and $\mathbf{b}_{\text{backcast}} \in \mathbb{R}^{F_{in}}$

are the learnable weight matrix and bias vector of the backcast convolution component.

Spatio-temporal Stack. An ST stack is the upper level of an ST block. More specifically, an ST stack consists of a sequence of L ST blocks that connect sequentially in a novel doubly residual structure (see Fig. 4c). This structure has two residual branches, one for the forecast of all the ST blocks in an ST stack and one for the backcast output of the previous ST block. The backcast output can be interpreted as the portion of the information that is not needed for the forecast job of the following ST blocks [43], making it easier for the following blocks to process the signal. The forecast output summarizes the final prediction of ST blocks, providing an implicit ensemble architecture, which demonstrates better performance compared to single models [4, 30]. Next, we describe the operation of the m -th ST stack in detail. As described before, the output of the l -th block in a stack is \mathcal{Y}_l and \mathcal{V}_l , as shown in Eqs. 18 and 19. Then, \mathcal{V}_l is used for computing the input of the next block, $X_{l+1}^\pi \in \mathbb{R}^{(N^\pi || \Delta N^\pi) \times F_{\text{in}} \times P}$ as shown in Eq. 20.

$$X_{l+1}^\pi = X_l^\pi - \mathcal{V}_l \quad (20)$$

The output of the last ST block in an ST stack (i.e., X_{L+1}) is also the residual for the next ST stack \tilde{X}_{m+1} , i.e., $\tilde{X}_{m+1} \equiv X_{L+1}$. The forecast residual $\tilde{\mathcal{Y}}_m \in \mathbb{R}^{(N^\pi || \Delta N^\pi) \times p \times H}$ is computed by summing the forecast output of all ST blocks in the ST stack as shown in Eq. 21.

$$\tilde{\mathcal{Y}}_m = \sum_{l=1}^L \mathcal{Y}_l \quad (21)$$

CAST Model. The entire CAST model consist of M ST stacks. As described before, the m -th ST stack produces two outputs, $\tilde{\mathcal{Y}}_m$ and \tilde{X}_m . The first output is the stack forecasting $\tilde{\mathcal{Y}}_i \in \mathbb{R}^{(N^\pi || \Delta N^\pi) \times p \times H}$, which contributes to the model forecast. The second output is the stack residual $\tilde{X}_m \in \mathbb{R}^{(N^\pi || \Delta N^\pi) \times F_{\text{in}} \times P}$, which is fed to the next stack. The global forecast $\hat{X} \in \mathbb{R}^{(N^\pi || \Delta N^\pi) \times p \times H}$ is the final output of the model, which is computed by aggregating the forecasts of all the stacks as follows.

$$\hat{X} = \frac{1}{M} \sum_{m=1}^M \tilde{\mathcal{Y}}_m \quad (22)$$

3.3 Continual Learning Module

The continual learning module is responsible for leveraging old knowledge, helping to reduce the training complexity caused by the evolution of RNs. First, we note that the motivation for reducing the complexity is to exploit the knowledge of the model trained on the historical RNs. With this knowledge, the model does not need to be trained on the entire RN subsequently. Rather, the model only needs to be trained on evolved parts. To achieve this, we propose two strategies. (i) The model does not need to be trained on the data of nodes, whose traffic patterns do not change much after an evolution. The model only uses a little data of these strongly unchanged nodes for consolidating the knowledge that is learned

from previous periods. (ii) The model has to be trained on the data of nodes whose traffic patterns are affected strongly by an evolution of the RN. The learned spatio-temporal representations of such nodes are no longer useful, so the new data from these nodes is used for updating the model. We proceed to introduce the continual learning module that adopts the proposed strategies. The core step is to select the nodes that are used for training. After the evolution of the RN, the continual learning module first picks all newly added and removed nodes. Assuming those old nodes close to added and removed nodes are strongly affected, the module also includes nodes adjacent to newly added and removed nodes. Simultaneously, the continual module addresses the impact of newly added and removed edges by selecting nodes adjacent to newly added and removed edges. The selected nodes and their edges are used for constructing the updated part ΔG . Then, the module identifies the most stable old nodes, which are used to revise the historical knowledge as an alternative to training on the entire RN. The most stable nodes are stored in a consolidation memory buffer \mathcal{B}_c , and this buffer is used to revise the model. The stability of a node is determined by the change in the data histogram on that node before and after the graph evolves. Specifically, we use the data from the last τ timestamps of period $\pi-1$ and of period π to produce the histogram at period $\pi-1$ (denoted as $H^{\pi-1}$) and at period π (denoted as H^π), respectively. After obtaining $H^{\pi-1}$ and H^π , we use the Earth mover's distance (EMD) to measure the stability of a node as shown in Eq. 23.

$$\text{EMD}(H^{\pi-1}, H^\pi) = \min_{Q \geq 0} \sum_i^{|H^{\pi-1}|} \sum_j^{|H^\pi|} Q_{i,j} \|H_i^{\pi-1} - H_j^\pi\| \quad (23)$$

Here, $Q \in \mathbb{R}^{|H^{\pi-1}| \times |H^\pi|}$ is the optimal transport plan matrix where $\sum_i Q_{i,j} = \frac{1}{|H^\pi|}$ and $\sum_j Q_{i,j} = \frac{1}{|H^{\pi-1}|}$. Intuitively, EMD is defined by a minimal transportation "cost" to convert histogram $H^{\pi-1}$ into histogram H^π . In simple terms, $H^{\pi-1}$ and H^π can be seen as representations of traffic flow distributions before and after an RN evolution, and the EMD is capable of quantifying the difference between these two distributions, even if there are significant shifts in the distributions (e.g., a highway is constructed nearby, which may lead to substantial traffic flow changes). The nodes with the lowest EMD are stable and can be used as rehearsal nodes for the traffic prediction model. The continual learning module selects the nodes with the lowest EMD and saves these most stable nodes in a consolidation memory buffer \mathcal{B}_c . Further, after evolving, there are unstable nodes (i.e., the patterns of these nodes have changed considerably). If the model keeps the old and now inaccurate space-time correlations of these nodes from the historical RN to conduct forecasting, the model's performance will decrease significantly. Therefore, it is necessary to select unstable nodes so that the model is forced to re-learn these nodes with new data. Similar to selecting the stable nodes with the lowest EMD, the continual learning module selects the nodes with the highest EMD and saves these unstable nodes in an update memory buffer \mathcal{B}_u , thereby treating these old nodes as newly added nodes.

We proceed to introduce Algorithm 1 that describes the training process that incorporates both the consolidation of historical

Algorithm 1: Training procedure

```

Input : model  $f_\theta^{\pi-1}$  trained on  $G^{\pi-1}$ , data  $X^\pi$  of  $G^\pi$ , data
 $X^{\pi-1}$  of  $G^{\pi-1}$ , period  $\tau$ , consolidation buffer size
 $|\mathcal{B}_c|$ , update buffer size  $|\mathcal{B}_u|$ 
Output: model  $f_\theta^\pi$ .
1  $EMD\_list = [ ]$ ;  $\mathcal{B}_u = [ ]$ ,  $\mathcal{B}_c = [ ]$ ;
2  $\Delta G^\pi \leftarrow G^\pi - G^{\pi-1}$ ;
3 for each node  $i \in V^{\pi-1}$  do
4    $\mathcal{E}^{\pi-1} \leftarrow$  Select last  $\tau$  timestamps from  $X^{\pi-1,i}$ ;
5    $\mathcal{E}^\pi \leftarrow$  Select last  $\tau$  timestamps from  $X^{\pi,i}$ ;
6    $H^{\pi-1} \leftarrow$  Build histogram for  $\mathcal{E}^{\pi-1}$ ;
7    $H^\pi \leftarrow$  Build histogram for  $\mathcal{E}^\pi$ ;
8    $div \leftarrow EMD(H^{\pi-1}, H^\pi)$ ;
9    $EMD\_list \leftarrow EMD\_list \cup (div)$ ;
10 sort( $EMD\_list$ );
11 Select  $|\mathcal{B}_c|$  nodes has lowest EMD and store to  $\mathcal{B}_c$ ;
12 Select  $|\mathcal{B}_u|$  nodes has highest EMD and store to  $\mathcal{B}_u$ ;
13  $f_\theta^\pi \leftarrow$  training  $f_\theta^{\pi-1}$  with  $\mathcal{B}_c$ ,  $\mathcal{B}_u$ , and  $\Delta G^\pi$ 

```

knowledge and the update of significantly altered nodes. Algorithm 1 takes the model $f_\theta^{\pi-1}$ trained on $G^{\pi-1}$, the data X^π of G^π , the data $X^{\pi-1}$ of $G^{\pi-1}$, the number of bins α , the number of timestamps τ , the consolidation buffer size $|\mathcal{B}_c|$, and the update buffer size $|\mathcal{B}_u|$ as the input to produce the model f_θ^π . First, the EMD_list and the two buffers, \mathcal{B}_c and \mathcal{B}_u , are initialized (line 1). Next, the algorithm constructs the graph from the evolved part of RN, denoted as ΔG^π , to update the model (line 2). Then, the historical and current data of each node in the historical RN $G^{\pi-1}$ are used to compute two data histograms (lines 3–7). The EMD is then computed based on the two obtained histograms (line 8). The divergence result is added to the EMD_list (line 9). After that, the EMD_list is sorted (line 10). Next, the consolidation buffer \mathcal{B}_c is filled with the nodes that have the lowest EMD, and the update buffer \mathcal{B}_u is filled with the nodes that have the highest EMD (lines 11–12). Finally, the model for period π , i.e., f_θ^π , is produced by training the model for period $\pi - 1$, i.e., $f_\theta^{\pi-1}$, on the data of evolved nodes and the buffer (line 13).

In summary, the continual learning module aims to capture updates and adapt to evolving RN data, hence avoiding complete re-training. The continual learning module only affects the training runtime and does not affect the real-time forecasting capability of traffic forecasting models. The forecasting runtime depends on the inference time of traffic forecasting models. After being updated and adapted, a model can forecast in real-time.

3.4 Objective Function

To train the proposed framework, we use the Huber loss [25] as the main objective function. This loss has demonstrated better results [14] for regression tasks than the traditional mean square error, especially when the training data contains noise and outliers that the Huber loss is less sensitive to. The main objective function is defined as follows.

$$\mathcal{L}_{\text{main}} = \begin{cases} \frac{1}{2} (\hat{X} - X)^2, & \text{if } |\hat{X} - X| \leq \delta \\ \delta \left(|\hat{X} - X| - \frac{1}{2} \delta \right), & \text{otherwise} \end{cases} \quad (24)$$

In addition, to better support the rehearsal algorithm that was described in the previous section, we apply the elastic weight consolidation method to the training procedure. This method measures the importance of each parameter θ_i in the parameter set θ . This allows us to estimate which parameters are the important ones for forecasting in the historical RN. Then, we can skip updating these parameters and focus on updating the less important parameters. By doing so, we achieve two benefits: (i) the model can reduce the knowledge-forgetting problem and (ii) the model can increase the ability to learn new knowledge. Elastic weight consolidation is a regularization method, which is defined as the objective function shown in Eq. 25.

$$\mathcal{L}_{\text{regularization}} = \sum_i F_i (\theta^\pi[i] - \theta^{\pi-1}[i])^2 \quad (25)$$

Here, $\theta^{\pi-1}[i]$ and $\theta^\pi[i]$ are the i -th parameter in the parameters set $\theta^{\pi-1}$ and θ^π , respectively. Next, F_i represents the importance of the i -th parameter in parameter set $\theta^{\pi-1}$ and is computed using the Fisher information [46] as follows.

$$F = \frac{1}{|\mathcal{X}^{\pi-1}|} \sum_{X^{\pi-1} \in \mathcal{X}^{\pi-1}} \frac{\partial \theta^{\pi-1}}{\partial X^{\pi-1}} \frac{\partial \theta^{\pi-1}^\top}{\partial X^{\pi-1}} \quad (26)$$

Here, $\frac{\partial \theta^{\pi-1}}{\partial X^{\pi-1}}$ is the partial first-order derivative of $\theta^{\pi-1}$ with respect to $X^{\pi-1}$. Finally, the overall objective function of the framework is the sum of the main objective function and the regularization.

$$\mathcal{L}_{\text{overall}} = \mathcal{L}_{\text{main}} + \lambda \mathcal{L}_{\text{regularization}}, \quad (27)$$

where λ is the hyperparameter to control the magnitude of the regularization.

3.5 Complexity Analysis

If a traffic forecasting model does not involve any recursive computation, most of its runtime is spent by the GNNs. To simplify the complexity term of the proposed framework, we consider the computational complexity mostly based on the number of nodes N^π of an RN G^π . We conduct a comparison between ordinary spatial embedding components, dynamic graph embedding methods, and our framework. When $\pi = 1$, the computational complexity is $O((N^\pi)^2)$ for each approach. When an RN evolves, i.e., $\pi > 1$, the need for a fully observed topological structure to extract spatial information becomes evident. Ordinary spatial embedding components require a re-initialization and a full training process with the data of all nodes in G^π , which increases the computational cost to $O((N^\pi)^2)$. Likewise, the computational cost of dynamic graph embedding methods is also $O((N^\pi)^2)$ since these have to train with the entire RN. However, dynamic graph embedding approaches offer a distinct advantage at traffic prediction due to their ability to model the evolution of the RN. In contrast, our proposed model's complexity is significantly lower, at $O((\Delta N^\pi + |\mathcal{B}_c \cup \mathcal{B}_u|)^2)$. Note

Table 1: Details of PEMS03-Evolve. $+|\cdot|$ and $-|\cdot|$ denote the number of added elements and removed elements, respectively.

Month	Apr	May	Jun	Jul	Aug	Sep	Oct
$ V $	655	715	768	822	834	850	871
$ E $	1,577	1,929	2,316	2,536	2,594	2,691	2,788
$+ \Delta V $	N/A	60	53	54	12	16	21
$+ \Delta E $	N/A	352	387	220	58	97	97
$- \Delta V $	N/A	26	14	7	16	25	25
$- \Delta E $	N/A	143	78	34	81	143	178
No. observations	8,856	8,856	8,856	8,856	8,856	8,856	8,856

Table 2: Details of PEMS04-Evolve. $+|\cdot|$ and $-|\cdot|$ denote the number of added elements and removed elements, respectively.

Month	Apr	May	Jun	Jul	Aug	Sep	Oct
$ V $	180	198	213	225	235	243	248
$ E $	308	400	469	535	583	614	623
$+ \Delta V $	N/A	18	15	12	10	8	5
$+ \Delta E $	N/A	92	69	66	48	31	9
$- \Delta V $	N/A	10	9	4	15	14	12
$- \Delta E $	N/A	36	30	24	72	66	60
No. observations	8,905	8,905	8,905	8,905	8,905	8,905	8,905

that $\Delta N^\pi \ll N^\pi$, so that $O((\Delta N^\pi + |\mathcal{B}_c \cup \mathcal{B}_u|)^2) \ll O((N^\pi)^2)$, where the total size of the buffers $|\mathcal{B}_c \cup \mathcal{B}_u|$ can be predetermined according to the specifications of the training system. As a result, the framework exhibits lower complexity than the other proposals when the network evolves (i.e. the framework only requires training on newly added nodes and selected old nodes), making it highly suitable for handling continuously evolving topologies.

4 EXPERIMENTS

4.1 Experimental Settings

Datasets. We experiment with two datasets: **PEMS03-Evolve** and **PEMS04-Evolve** that are collected in seven *periods*. For simplicity, we consider periods π with a duration of a *month*. Obviously, if we change the duration, e.g., to *quarters*, the proposal is not affected. Every month, a new RN is created by adding and removing nodes and edges to and from the RN of the previous month. The data is collected from metropolitan areas of California by California Transportation Agencies Performance Measurement System (PEMS)¹. Details of the datasets are provided in Tables 1 and 2. Every 5 minutes, a time series observation is generated, representing the average traffic flow without revealing any personally identifiable information such as vehicle identity. Following existing studies, we split all datasets with a ratio 60%:20%:20% into training sets, validation sets, and testing sets, respectively.

Forecasting Setting. We design two scenarios to evaluate the accuracy and the runtime of the proposed framework. In the first scenario, we evaluate CAST. CAST is first trained by using the data of the first period. Then, CAST is reinitialized and fully trained every period by using the data of all nodes for that period. Also, all the other baselines are evaluated in this setting. We report the accuracy of the framework for the last period and the runtime for the whole training procedure. The first scenario mimics the trivial solution of training a traffic forecasting model each time an RN evolves.

In the second scenario, we evaluate TEAM, which is first trained by using the data of the first period. Then, the framework is transferred and partially trained every following period by using only the data of newly updated nodes and the data in the buffer, i.e., employing

the continual learning module. We also report the accuracy of the framework for the last period and the runtime for the whole training procedure. Intuitively, the last period is the most difficult setting because the topology of the last period is the most different from that of the first period. By evaluating the accuracy of the model in the last period, we aim to evaluate the model in the most challenging setting. If the model performs well here, it can also perform well in intermediate periods. We follow existing studies for the setting of forecasting horizon [35, 59, 60]. Given the previous $P = 12$ time steps (i.e., 1 hour), we aim to forecast the next $K = 12$ time steps (i.e., 1 hour).

Baselines. We compare our framework with the following baselines: (1) HA [26]: a smoothing method, which forecasts the future values by averaging historical values; (2) VAR [22]: an auto-regressive method, which assumes the data follow a predefined function; (3) SVR [6]: a kernel-based method, which maps temporal data into a latent space and uses support vectors for forecasting; (4) GRU [11]: a pure RNN model, which can capture long-term dependencies using gate mechanism; (5) DCRNN [35]: a sequence-to-sequence architecture, which uses GCNs to model spatial information and RNN to model temporal information; (6) STGCN [59]: a sandwich architecture, which encloses GCNs with 1DCNNs; (7) GWN [54]: a causal CNN based method, which uses GCNs to model spatial information and dilated causal CNNs to model temporal information; (8) MSTGCN [21]: a state-of-the-art method, which incorporates multi-view mechanism into GCNs. (9) ASTGCN [21]: a state-of-the-art method, which uses attentions and GCNs for model temporal and spatial information, respectively; (10) STSGCN [47]: an advanced approach that utilizes localized a spatial-temporal subgraph module to capture the spatial-temporal correlation simultaneously. (11) EvolveGCN [44]: a state-of-the-art dynamic graph embedding method that employs RNNs and GCNs to capture change between graph snapshots; (12) DyRep [49]: a state-of-the-art dynamic graph embedding method that models the local and global topological evolution; (13) GMAN [60]: a traffic forecasting framework using multiple graph attention networks to model spatio-temporal dynamics; (14) EnhanceNet [13]: a plugin that integrates to RNNs and GCNs to capture correlation among different entities; (15) ST-WA [14]: a framework that considers location-specific and time-varying model parameters to capture complex spatio-temporal dynamics; (16) D2STGNN [45]: a framework that captures the diffusion and inherent traffic information separately; (17) PDFFormer [27]: a model that captures both short-range and long-range dynamic spatial dependencies; (18) TrafficStream [8]: a method to efficiently support traffic forecasting on expandable RNs. To adapt EvolveGCN and DyRep to traffic forecasting, we add a 1DCNN layer on top of both methods to produce the forecasting results. Most of the baselines are applicable only to the first scenario whereas TrafficStream is also applicable to the second scenario.

Evaluation Metrics. We use three metrics including Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE) [35] between forecasted time series and ground truth to measure the accuracy. We also report the average runtime for each epoch and the total runtime. The accuracy metrics are defined as follows.

¹<http://pems.dot.ca.gov/>



Figure 6: PEMS03-Evolve RN visualizations. Red nodes denote added nodes and blue nodes denote removed nodes, respectively.



Figure 7: PEMS04-Evolve RN visualizations. Red nodes denote added nodes and blue nodes denote removed nodes, respectively.

$$MAE = \frac{1}{K} \sum_{t=1}^K \left| \mathbf{X}_{T+t}^\pi - \hat{\mathbf{X}}_{T+t}^\pi \right| \quad (28)$$

$$RMSE = \sqrt{\frac{1}{K} \sum_{t=1}^K \left(\mathbf{X}_{T+t}^\pi - \hat{\mathbf{X}}_{T+t}^\pi \right)^2} \quad (29)$$

$$MAPE = \frac{1}{K} \sum_{t=1}^K \left| \frac{\hat{\mathbf{X}}_{T+t}^\pi - \mathbf{X}_{T+t}^\pi}{\mathbf{X}_{T+t}^\pi} \right| \cdot 100\% \quad (30)$$

Hyperparameter Settings. We train the framework using the Adam optimizer with a learning rate of 0.001 and a batch size of 64. The total number of epochs is set to 200, and we use early stopping with a patience of 15. The regularization factor λ (see Eq. 27) is set to 0.0001. We tune the other hyperparameters by random search on the validation data as follows. We consider different hyperparameter settings and report the best result on the validation data for all methods. Specifically, we define a range for each hyperparameter. We then use a random search with 100 random combinations to explore the hyperparameter space and identify a hyperparameter setting that gives the best result on the validation data among all the explored hyperparameter settings. We then report this best result and use this hyperparameter setting as the default setting. Next, we study the sensitivity of different hyperparameters. To do so, we vary a chosen hyperparameter in its range while fixing the other hyperparameters to their default settings. We proceed to provide the ranges for the hyperparameters.

For the proposed framework, we vary the number of ST blocks L , the number of ST stacks M , and the number of head U in the multi-head attention among 1, 2, 3, 4, and 5. We vary the number of convolution filters among 16, 32, 64, 128, and 256. We vary the consolidation buffer size $|\mathcal{B}_c|$ and update buffer size $|\mathcal{B}_u|$ among 5%, 10%, 15%, 20%, 25%, and 30% of the number of nodes N^π . For all the methods that involve RNN (i.e., GRU, DCRNN, and EvolveGCN), we vary the number of the hidden units among 16, 32, 64, 128, and 256, and the number of hidden layers among 1, 2, 3, 4, and 5. For all the methods that involve CNN (i.e., STGCN, GWN, MSTGCN, and ASTGCN), we vary the number of convolution filters among 16, 32, 64, 128, and 256, and the number of hidden layers among 1, 2, 3, 4, and 5. For all methods, we vary the number of GCN filters among 16, 32, 64, 128, and 256.

Implementation Details. We implement the proposed framework and other baselines on Python 3.7, PyTorch 1.10, Geometric 2.0.4, and Sklearn 0.24. All experiments are conducted on a cluster server, which runs Linux Ubuntu 18.04.6 LTS. The server is equipped with a ten-core Intel(R) Xeon(R) W-2155 CPU, 128 GBs RAM, and two GPUs Titan RTX each with 24 GBs VRAM.

4.2 Experimental Results

Main Results. Tables 3 and 4 show the overall accuracy and runtime of the proposed framework and the baselines. In the first scenario, CAST achieves the best accuracy on both datasets. The only exception is that CAST is behind DCRNN in the short-term setting (i.e., 15 mins) on **PEMS03-Evolve**. In terms of runtime, CAST is efficient and only is behind GRU and STGCN. The accuracy of PDFormer is also high, and it is the runner-up on both datasets. However, PDFormer is very inefficient: it is the slowest baseline and is 12x slower than CAST. In the second scenario, TEAM can maintain good accuracy on **PEMS03-Evolve**. Further, on **PEMS04-Evolve**, TEAM is strongly competitive in terms of accuracy. Considering runtime, TEAM exhibits very good efficiency, especially in the continual setting. TrafficStream also exhibits very good efficiency. However, its accuracy is well below that of TEAM. On both datasets, TEAM is only slower than GRU, which does not perform any spatial computation. In summary, the result indicates that CAST can outperform the baselines w.r.t. accuracy in the first scenario and that TEAM outperform the baselines w.r.t. runtime while maintaining competitive accuracy in the second scenario.

Performance Improvement Significance. To evaluate the hypothesis that whether the performance improvements of the proposed framework over the baselines are statistically significant, we conduct t -tests to assess the significance of the proposed framework against the baselines on the average results of all datasets. The p -values for all the metrics are below 0.005. This indicates that the performance improvements over the state-of-the-art methods are statistically significant.

Ablation Study. We study the effect of each component in the proposed framework by removing the convolution (denoted as TEAM-CONV), removing the attention (denoted as TEAM-ATT), swapping the convolution and the attention position, i.e., the input is fed into the convolution first, then the output of the convolution is fed to the attention (denoted as TEAM-SWAP), removing the backcast convolution (denoted as TEAM-BACK), removing the stack residual (see Eq. 22), i.e., the output of the entire model only is the output of

Table 3: Overall accuracy and runtime, PEMS03-Evolve.

Scenario	Model	15 mins			30 mins			60 mins			Runtime (seconds)	
		MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE	Total	Average
1st	HA	17.25	31.55	25.67	17.14	31.34	25.67	17.19	31.49	25.43	89.91	-
	VAR	17.65	28.41	23.48	18.49	29.28	23.63	20.93	33.03	26.00	502.81	-
	SVR	15.04	24.37	17.98	16.21	26.10	21.23	19.51	32.36	25.68	N/A	N/A
	GRU	13.49	23.04	18.83	14.58	25.07	20.58	17.34	29.02	24.92	6316.75	16.21
	DCRNN	11.97	18.57	19.82	14.77	34.09	22.93	17.28	29.50	24.13	231162.3	426.45
	STGCN	12.41	20.34	16.89	14.43	23.63	20.69	17.58	28.63	25.71	22848.19	54.60
	STGCN	13.15	21.09	17.35	13.48	22.05	17.84	14.62	24.02	19.54	63687.41	163.05
	GWN	13.10	21.67	17.17	14.02	23.45	18.86	16.14	26.50	21.14	42294.72	96.77
	MSTGCN	13.67	21.73	19.73	14.55	23.65	20.59	16.84	27.34	24.98	29075.06	61.87
	ASTGCN	13.28	21.66	20.64	14.67	23.60	21.73	16.78	27.29	24.82	37941.11	63.66
	EvoiveGCN	14.41	23.56	20.01	15.73	25.94	21.98	18.48	30.98	25.83	34936.41	48.51
	DyRep	13.65	22.26	21.48	15.11	23.61	21.79	17.22	27.21	24.89	39486.63	110.59
	GMAN	18.13	29.40	24.42	20.73	30.82	24.79	23.16	31.99	26.28	158859.4	229.33
	EnhanceNet	12.47	21.54	18.98	13.42	22.19	20.60	15.04	24.72	22.33	67431.15	94.45
	ST-WA	12.57	20.89	20.05	13.70	23.21	23.82	14.45	23.85	24.75	95898.2	248.31
	D2STGNN	14.47	24.06	19.52	16.23	26.79	21.89	17.24	29.72	27.02	85794.14	205.08
	PDFormer	12.49	20.43	17.47	<u>13.26</u>	<u>21.90</u>	18.75	<u>14.28</u>	<u>23.22</u>	19.83	331641.3	736.85
	CAST (ours)	12.16	20.28	17.85	13.09	21.63	<u>18.63</u>	14.17	22.91	19.14	27341.47	61.08
2nd	TrafficStream	13.74	22.86	21.44	15.21	25.00	20.92	17.72	29.40	22.41	6974.98	36.12
	TEAM (ours)	12.82	21.37	17.98	13.57	22.89	18.95	15.16	25.69	21.88	6574.24	33.55

Table 4: Overall accuracy and runtime, PEMS04-Evolve.

Scenario	Model	15 mins			30 mins			60 mins			Runtime (seconds)	
		MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE	Total	Average
1st	HA	2.21	4.75	4.79	2.21	4.75	4.79	2.21	4.75	4.79	57.45	-
	VAR	1.96	3.79	3.82	2.62	4.58	4.98	2.90	5.07	5.66	206.72	-
	SVR	1.79	3.45	3.48	2.08	4.35	4.48	2.57	5.67	5.45	N/A	N/A
	GRU	4.33	6.47	9.51	5.60	6.61	9.89	6.45	7.36	9.49	1784.52	8.61
	DCRNN	1.51	2.92	2.87	1.78	3.76	3.41	2.42	4.47	4.19	66014.6	119.30
	STGCN	2.79	5.06	5.89	2.86	5.17	4.96	3.15	5.54	6.75	6160.2	14.76
	STSGCN	2.86	5.87	6.55	2.99	6.16	6.95	3.25	6.69	7.68	29596.6	70.44
	GWN	1.37	2.90	2.74	1.71	3.89	3.56	2.07	4.58	4.43	12025.85	26.43
	MSTGCN	1.54	3.06	3.05	2.01	4.12	4.23	2.69	5.36	5.95	8436.25	18.64
	ASTGCN	1.64	3.49	3.53	1.98	4.20	4.27	2.37	4.98	5.45	10818.8	23.86
	EvoiveGCN	1.50	2.88	3.17	1.75	3.56	3.75	2.12	4.52	4.67	9183.63	15.92
	DyRep	1.52	3.01	3.13	1.84	3.81	3.67	2.14	4.51	4.74	13239.52	32.23
	GMAN	2.28	4.40	4.80	2.96	5.83	6.52	2.97	5.85	6.60	47048.81	67.31
	EnhanceNet	1.40	2.87	2.80	1.75	3.90	3.75	2.12	4.74	4.70	16859.54	24.08
	ST-WA	1.61	3.45	3.44	1.83	4.11	4.02	2.11	4.66	4.63	26777.4	78.68
	D2STGNN	1.53	3.13	3.20	2.10	4.61	4.78	2.61	6.04	6.43	22540.51	64.43
	PDFormer	1.30	2.62	2.53	1.57	3.26	3.17	1.97	4.22	4.22	113412.2	233.33
	CAST (ours)	1.28	2.50	2.51	1.57	3.20	3.19	1.96	4.10	4.14	7962.96	17.82
2nd	TrafficStream	1.68	2.99	3.68	2.15	3.93	4.35	2.66	4.76	5.56	2250.41	10.24
	TEAM (ours)	1.37	2.66	2.61	1.60	3.29	3.31	2.05	4.21	4.25	2007.21	9.11

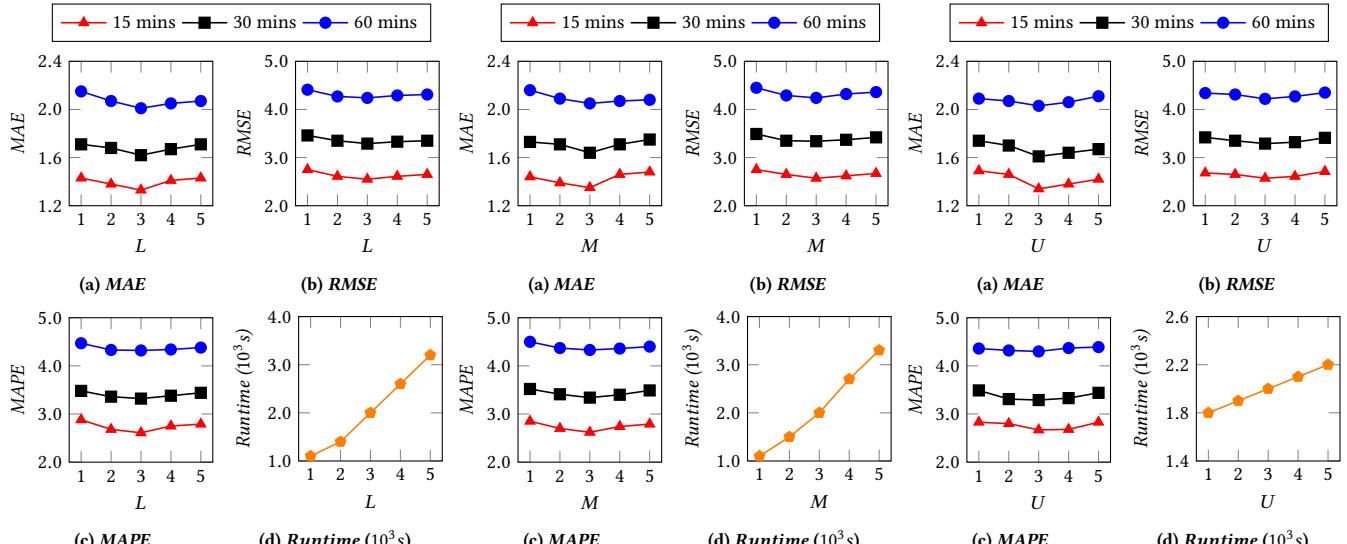


Figure 8: Effect of L , PEMS04-Evolve.

the last ST stack (denoted as TEAM-RES), and remove the continual module (denoted as TEAM-CONT). All the variants perform in the

Figure 9: Effect of M , PEMS04-Evolve.

second scenario. We report findings on dataset **PEMS04-Evolve** only. The results on **PEMS03-Evolve** show similar trends. Table 5

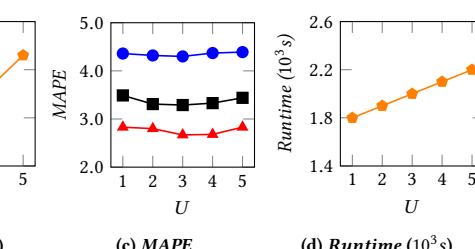


Figure 10: Effect of U , PEMS04-Evolve.

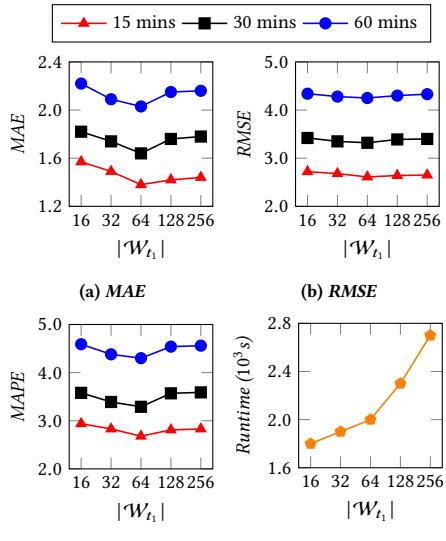


Figure 11: Effect of $|W_{t_1}|$, PEMSO4-Evolve.

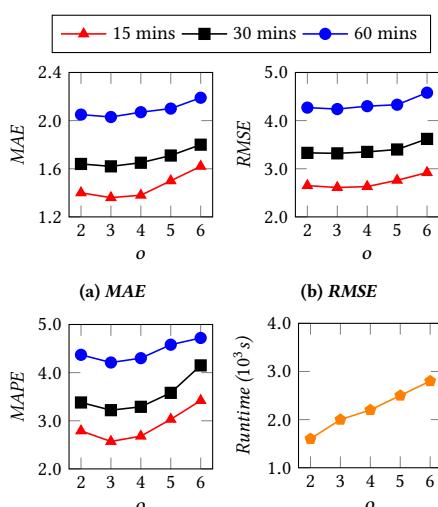


Figure 12: Effect of o , PEMSO4-Evolve.

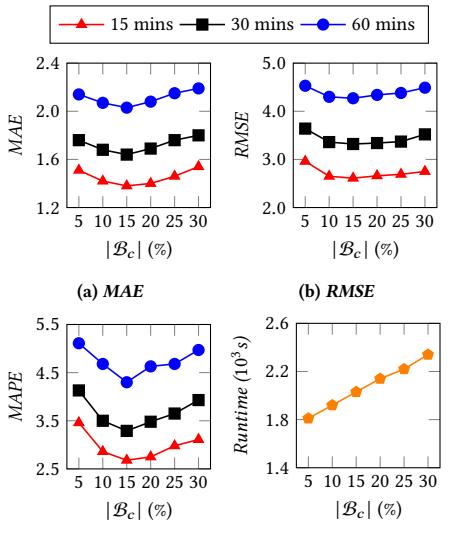


Figure 13: Effect of $|\mathcal{B}_c|$, PEMSO4-Evolve.

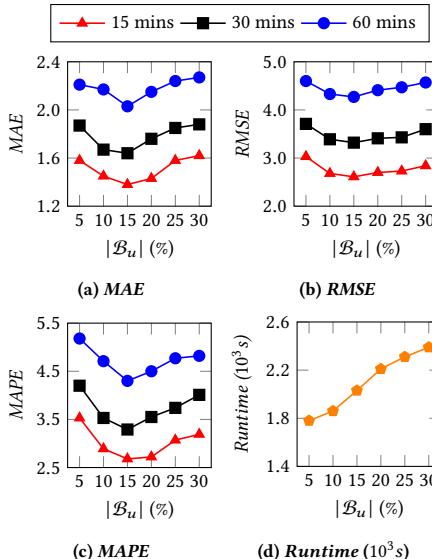


Figure 14: Effect of $|\mathcal{B}_u|$, PEMSO4-Evolve.

Table 5: Ablation study, PEMSO4-Evolve.

Model	15 mins			30 mins			60 mins		
	MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE
TEAM-CONV	1.49	2.74	2.92	1.71	3.39	3.47	2.07	4.28	4.45
TEAM-ATT	1.83	3.06	3.17	1.93	3.74	3.98	2.28	4.92	5.25
TEAM-SWAP	1.50	2.93	3.42	1.79	3.64	4.16	2.17	4.59	5.29
TEAM-BACK	1.53	2.98	3.40	1.78	3.62	4.06	2.19	4.52	5.13
TEAM-RES	1.56	2.73	2.92	1.81	3.43	3.56	2.14	4.34	4.45
TEAM-CONT	4.71	8.71	11.45	4.74	8.79	11.57	4.94	8.87	11.64
TEAM	1.37	2.66	2.61	1.60	3.29	3.31	2.05	4.21	4.25

shows the results w.r.t. MAE, RMSE, and MAPE. We do not include the runtime in Table 5 because the runtime of TEAM variants are only slightly different from the entire framework. The results show that TEAM-CONT performs poorly and becomes the worst variant. This suggests that the continual learning module plays an important role in the proposed framework to enable the ability to train TEAM on evolved parts of RNs. Both TEAM-ATT and TEAM-SWAP

Table 6: Number of evolved nodes vs. runtime, TEAM, PEMSO4-Evolve.

#Evolved nodes	#Training nodes	Prep. runtime (seconds)	Total runtime (seconds)	Avg. runtime (seconds)
13	24	1.7	134.8	6.74
16	36	1.7	144.2	7.21
19	45	1.8	157.4	7.87
22	51	1.9	159.6	7.98
25	63	1.9	163.4	8.17
28	70	2.1	166.8	8.34

perform worse than TEAM. This suggests that the combination of convolution and attention improves the accuracy on incremental time series. Next, TEAM-SWAP, TEAM-BACK, and TEAM-RES perform worse than TEAM. This suggests that our doubly residual design and the proposed architecture of TEAM are efficient.

Empirical Complexity Study. Adding to the complexity analysis in Section 3.5, we study the complexity empirically by considering

Table 7: Accuracy during immediate periods, PEMS04-Evolve.

Scenario	Model	Metric	Apr	May	Jun	Jul	Aug	Sep	Oct
1st	EnhanceNet	MAE	1.33	1.38	1.48	1.51	1.64	1.68	1.76
		RMSE	2.72	2.79	2.83	2.92	2.95	3.18	3.84
		MAPE	2.58	2.67	2.61	2.64	2.88	3.26	3.75
2nd	CAST	MAE	1.31	1.34	1.46	1.47	1.56	1.59	1.60
		RMSE	2.68	2.66	2.81	3.01	3.09	3.27	3.27
		MAPE	2.49	2.51	2.97	2.91	3.22	3.29	3.28
2nd	TrafficStream	MAE	1.56	1.66	1.67	1.75	1.76	1.92	2.16
		RMSE	3.24	3.31	3.39	3.44	3.74	3.75	3.89
		MAPE	3.19	3.27	3.46	3.51	3.63	3.94	4.53
2nd	TEAM	MAE	1.31	1.45	1.52	1.54	1.60	1.67	1.67
		RMSE	2.68	2.81	3.04	3.28	3.27	3.36	3.38
		MAPE	2.49	2.87	2.98	3.33	3.57	3.36	3.39

Table 8: Accuracy for stable and unstable nodes, PEMS04-Evolve.

Scenario	Model	Nodes	Metric	May	Jun	Jul	Aug	Sep	Oct
1st	EnhanceNet	Stable	MAE	1.65	1.84	2.16	2.16	1.67	1.90
			RMSE	3.78	4.20	4.13	4.92	3.92	3.95
			MAPE	3.51	4.08	4.42	5.57	3.93	3.83
		Unstable	MAE	1.37	2.14	1.71	1.17	1.34	1.49
			RMSE	2.95	4.82	3.53	3.34	2.62	2.85
			MAPE	2.29	2.34	2.94	2.25	2.95	3.62
2nd	CAST	Stable	MAE	0.34	0.28	1.08	0.19	1.01	1.36
			RMSE	2.25	1.18	2.89	1.03	1.94	2.85
			MAPE	1.13	0.51	2.34	0.31	1.63	2.39
		Unstable	MAE	0.83	1.41	1.12	1.16	1.21	1.64
			RMSE	1.61	3.00	2.32	2.09	3.46	3.31
			MAPE	1.31	2.89	1.92	1.88	3.14	2.94
2nd	Traffic Stream	Stable	MAE	1.41	1.31	1.66	1.37	1.79	2.01
			RMSE	2.44	3.05	3.63	2.16	3.11	4.17
			MAPE	2.29	2.34	2.94	2.25	2.95	3.63
		Unstable	MAE	4.33	2.31	2.39	1.66	3.41	2.59
			RMSE	9.79	4.73	5.34	2.98	6.72	4.98
			MAPE	13.19	4.69	5.01	2.75	9.27	4.78
2nd	TEAM	Stable	MAE	0.82	0.84	1.29	0.99	0.91	0.85
			RMSE	1.61	1.84	2.44	2.11	2.94	1.14
			MAPE	1.31	1.41	2.18	1.68	3.27	1.31
		Unstable	MAE	1.88	1.55	1.81	1.20	0.99	1.40
			RMSE	4.32	3.21	3.44	2.13	1.98	2.84
			MAPE	4.26	3.29	3.29	1.96	1.61	2.48

Table 9: Effect of sampling strategy, PEMS04-Evolve.

Sampling	15 min			30 min			60 min		
	MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE
Random	2.66	3.86	4.74	2.88	4.31	5.23	3.20	4.98	6.01
Random Walk	2.61	3.74	4.56	2.77	4.20	5.14	3.16	4.89	5.88
Degree	1.59	2.81	3.07	1.83	3.40	3.62	2.24	4.24	4.57
Closeness	1.89	3.10	3.59	2.25	3.66	4.06	2.85	4.65	5.15
Betweenness	1.98	3.11	3.65	2.30	3.74	4.13	2.92	4.69	5.22
PageRank	1.78	2.89	3.15	1.96	3.55	3.71	2.31	4.29	4.75
TEAM	1.37	2.66	2.61	1.60	3.29	3.31	2.05	4.21	4.25

Table 10: Accuracy and avg. runtime (s) vs. update frequencies, PEMS04-Evolve.

Scenario	Model	Metric	Day	Week	Month	Quarter	Year
1st	EnhanceNet	MAE	11.51	3.44	1.76	1.92	2.05
		RMSE	15.28	6.35	3.84	4.07	4.17
		MAPE	51.83	8.63	3.75	3.77	3.79
		Avg. RT	4.23	10.66	24.08	75.47	305.07
2nd	CAST	MAE	11.41	2.86	1.60	1.79	1.95
		RMSE	15.14	4.97	3.27	3.30	3.41
		MAPE	51.56	7.17	3.28	3.46	3.60
		Avg. RT	3.07	6.92	17.82	56.41	227.26
2nd	Traffic Stream	MAE	11.72	3.16	2.16	2.21	2.38
		RMSE	15.78	5.34	3.89	4.05	4.11
		MAPE	55.13	8.61	4.53	4.66	4.79
		Avg. RT	2.28	4.21	10.24	23.43	73.18
2nd	TEAM	MAE	10.19	2.82	1.67	1.85	1.99
		RMSE	13.99	4.85	3.38	3.59	3.64
		MAPE	47.45	6.72	3.39	3.44	3.69
		Avg. RT	1.99	3.51	9.11	18.90	57.49

the effect of the number of evolved nodes on the training time. We thus adjust the number of added and removed nodes and observe the total training time. We set the updating and the consolidation

buffer size to 10. We also report the preprocessing time of the continual learning module (see lines 1–12 in Algorithm 1). The results on **PEMS04-Evolve** are presented in Table 6. Similar trends are observed on **PEMS03-Evolve**. The results show that the number of trained nodes is roughly two to three times the number of evolved nodes. Next, the training time is low compared to the baselines (see Table 4). This is evidence of the effectiveness of the continual learning module at reducing the training time by focusing only on essential nodes. The results also show that the preprocessing time of the continual learning module is low, indicating that the additional overhead of the continual learning module is negligible.

Accuracy during Immediate Periods. In addition to reporting on the accuracy for the last period as in Tables 3 and 4, we report on the accuracy during immediate periods (i.e., accuracy across all periods). We adopt the same two scenarios as for the main result. For brevity, we only report the average result across three horizons of our proposal and EnhanceNet and TrafficStream on **PEMS04-Evolve**. The results for the other baselines on both **PEMS03-Evolve** and **PEMS04-Evolve** share the same characteristics. The results in Table 7 show that CAST always outperforms the EnhanceNet in the first scenario and that TEAM always outperforms TrafficStream in the second scenario. TEAM is only insignificantly behind EnhanceNet in May, June, July, and August, and TEAM outperforms EnhanceNet in the first and the two last months. This is because TEAM can learn from historical periods and use the resulting knowledge to improve forecasting accuracy in future periods.

Accuracy for Stable and Unstable Nodes. In addition to reporting on the accuracy across all nodes as in Tables 3 and 4, we report on the accuracy for only stable and unstable nodes. Specifically, we report on the accuracy for nodes in the buffers $|\mathcal{B}_c|$ and $|\mathcal{B}_u|$. We again adopt the two scenarios from the main results. We only report average results across three horizons of our proposal and EnhanceNet and TrafficStream on **PEMS04-Evolve**. The results for the other baselines on both datasets share the same characteristics. Table 8 shows that CAST outperforms EnhanceNet w.r.t. accuracy for both stable and unstable nodes in the first scenario. Similarly, TEAM outperforms TrafficStream w.r.t. accuracy for both stable and unstable nodes in the second scenario. Moreover, TEAM outperforms CAST in the two last periods. This is because TEAM can learn from historical periods and use the resulting knowledge to improve forecasting accuracy in future periods.

Effect of Sampling Strategy. In addition to reporting on the accuracy using the proposed sampling strategy for \mathcal{B}_c and \mathcal{B}_u (see Algorithm 1), we report on the accuracy for other sampling strategies. (1) We randomly select [34] $|\mathcal{B}_c|$ and $|\mathcal{B}_u|$ nodes for \mathcal{B}_c and \mathcal{B}_u , respectively; (2) We use random walks [34] to select $|\mathcal{B}_c|$ and $|\mathcal{B}_u|$ nodes for \mathcal{B}_c and \mathcal{B}_u , respectively; (3) We use a degree matrix [3] to select $|\mathcal{B}_c|$ nodes with highest degree and $|\mathcal{B}_u|$ nodes with lowest degree for \mathcal{B}_c and \mathcal{B}_u , respectively; (4) We use the closeness centrality metric [3] to select $|\mathcal{B}_c|$ nodes with highest centrality and $|\mathcal{B}_u|$ nodes with lowest centrality for \mathcal{B}_c and \mathcal{B}_u , respectively; (5) We use the betweenness centrality metric [3] to select $|\mathcal{B}_c|$ nodes with highest centrality and $|\mathcal{B}_u|$ nodes with lowest centrality for \mathcal{B}_c and \mathcal{B}_u , respectively; (6) We use PageRank [3] to select $|\mathcal{B}_c|$

nodes with highest rank and $|\mathcal{B}_u|$ nodes with lowest rank for \mathcal{B}_c and \mathcal{B}_u , respectively. For brevity, we only report the result for the last month on **PEMS04-Evolve**. The result on **PEMS03-Evolve** shares the same characteristics. Table 9 shows that the random sampling and random walk strategies perform the worst and that the sampling strategies based on degree, centrality, and PageRank perform better. This suggests that the latter three strategies can be used to select stable and unstable nodes. Our proposal achieves the best accuracy. This suggests that using temporal information (i.e., τ), as does our strategy, is important.

Effect of Evolving Frequency. We study the effect of evolving frequency π . Thus, in addition to capturing the evolution of RNs every month (i.e., $\pi = 1$ month) as in the default setting, we capture the evolution of RNs every day, every week, every quarter, and every year. We synthesize evolved RNs by randomly adding and removing 10–15% of the nodes and 20–25% of the edges every period. We only report the average result across three horizons for the last period (i.e., the 7-th day, 7-th week, the 7-th quarter, and the 7-th year) on **PEMS04-Evolve**. The results on **PEMS03-Evolve** share similar characteristics. Table 10 shows that TEAM achieves the best accuracy compared to the baselines when the RNs evolve rapidly ($\pi \leq 1$ week). This is because TEAM can leverage knowledge extracted from previous periods, mitigating the impact of limited training samples (e.g., 288 samples for a day and 1440 samples for a week) that reduce the performance of the other methods that reinitialize and retrain for every period. When RNs evolve more slowly, e.g., $\pi \geq 1$ month, the methods that are restricted to the first scenario perform better than TEAM due to having sufficient training samples, and CAST outperforms the other baselines w.r.t accuracy. The results exhibit the efficiency and accuracy of TEAM when learning with limited training samples and adapting to rapidly evolving RNs.

Effect of the Number of ST Blocks L . We study the effect of the number of ST blocks L in the proposed framework. We vary L among 1, 2, 3, 4, and 5. Due to the space limitation, we report on the effect of the number of ST block L on dataset **PEMS04-Evolve** only. The results on **PEMS03-Evolve** show similar trends. Fig. 8 shows the results w.r.t. MAE, RMSE, MAPE, and runtime. The proposed framework achieves the best performance when $L = 3$, which is the default value. When $L < 3$, the framework has not learned a good representation due to the underfitting. When $L > 3$, the framework consists of more parameters, which may be easier to be overfitting or to be trapped in the local optimum. When $L = 3$, the best trade-off is achieved.

Effect of the Number of ST Stacks M . We study the effect of the number of ST stacks M in the proposed framework. In particular, we vary M among 1, 2, 3, 4, and 5. Due to the space limitation, we report on the effect of the number of ST stacks M on dataset **PEMS04-Evolve** only. The results on **PEMS03-Evolve** show similar trends. Fig. 9 shows the results w.r.t. MAE, RMSE, MAPE, and runtime. The proposed framework achieves the best performance when $M = 3$, which is the default value. When $M < 3$, the framework has not learned a good representation due to the underfitting. When $M > 3$, the framework consists of more parameters, which may be easier to

be overfitting or to be trapped in the local optimum. When $M = 3$, the best trade-off is achieved.

Effect of the Number of Attention Heads U . We study the effect of the number of head U in the multi-head attention (see Eq. 12). In particular, we vary U among 1, 2, 3, 4, and 5. Due to the space limitation, we report on the effect of the number of ST stacks M on dataset **PEMS04-Evolve** only. The results on **PEMS03-Evolve** show similar trends. Fig. 10 shows the results w.r.t. MAE, RMSE, MAPE, and runtime. The proposed framework achieves the best performance when $U = 3$, which is the default value. When $U < 3$, the framework has not learned a good representation due to the underfitting. When $U > 3$, the framework consists of more parameters, which may be easier to be overfitting or to be trapped in the local optimum. When $U = 3$, the best trade-off is achieved.

Effect of the Number of Temporal Convolution Filters. We study the effect of the number of temporal convolution filters $|\mathcal{W}_{t_1}|$ in the filter bank \mathcal{W}_{t_1} . In particular, we vary $|\mathcal{W}_{t_1}|$ among 16, 32, 64, 128, and 256. Due to the space limitation, we report on the effect of the number of convolution filters $|\mathcal{W}_{t_1}|$ on dataset **PEMS04-Evolve** only. The results on **PEMS03-Evolve** show similar trends. Fig. 11 shows the results w.r.t. MAE, RMSE, MAPE, and runtime. The proposed framework achieves the best performance when $|\mathcal{W}_{t_1}| = 64$, which is the default value. When $|\mathcal{W}_{t_1}| < 64$, the framework has not learned a good representation due to the underfitting. When $|\mathcal{W}_{t_1}| > 64$, the framework consists of more parameters, which may be easier to be overfitting or to be trapped in the local optimum. When $|\mathcal{W}_{t_1}| = 64$, the best trade-off is achieved.

Effect of the Order o in Chebyshev Function. We study the effect of the order o in Chebyshev function (see Eq. 7). In particular, we vary o among 2, 3, 4, 5, and 6. Due to the space limitation, we report on the effect of the order o on dataset **PEMS04-Evolve** only. The results on **PEMS03-Evolve** show similar trends. Fig. 12 shows the results w.r.t. MAE, RMSE, MAPE, and runtime. The proposed framework achieves the best performance when $o = 3$, which is the default value. When $o < 3$, the framework has not learned a good representation due to the underfitting. When $o > 3$, the framework consists of more computations, which may be easier to be overfitting or to be trapped in the local optimum. When $o = 3$, the best trade-off is achieved.

Effect of the Buffer Size $|\mathcal{B}_c|$ and $|\mathcal{B}_u|$. We study the effect of the consolidation buffer size $|\mathcal{B}_c|$ and update buffer size $|\mathcal{B}_u|$. In particular, we vary one of $|\mathcal{B}_c|$ and $|\mathcal{B}_u|$ among 5%, 10%, 15%, 20%, 25%, and 30%, while fixing the value at 15% for the other. Due to the space limitation, we report on the effect of the consolidation buffer size $|\mathcal{B}_c|$ and update buffer size $|\mathcal{B}_u|$ on dataset **PEMS04-Evolve** only. The results on **PEMS03-Evolve** show similar trends. Fig. 13 and Fig. 14 show the results w.r.t. MAE, RMSE, MAPE, and runtime when varying $|\mathcal{B}_c|$ and $|\mathcal{B}_u|$, respectively. The proposed framework achieves the best performance when $|\mathcal{B}_c| = |\mathcal{B}_u| = 15\%$, which is the default value. When $|\mathcal{B}_c| < 15\%$, the framework lacks enough historical data for rehearsal and almost trains only on the data of newly added nodes and thus cannot exploit the continual module for revisiting the learned knowledge, which yields sub-optimal performance. When $|\mathcal{B}_c| > 15\%$, the framework exploits too much historical knowledge from historical nodes and fails to learn the

information from newly added nodes, which also yields sub-optimal performance. When $|\mathcal{B}_u| < 15\%$, the model may not update nodes that have different patterns, thus preserving inaccurate knowledge, causing substandard performance. When $|\mathcal{B}_u| > 15\%$, the model requires frequent updates to its knowledge, which may conflict with previously learned historical knowledge, leading to deficient outcomes. When $|\mathcal{B}_c| = |\mathcal{B}_u| = 15\%$, the best trade-off is achieved.

Effect of the Length of Sampling Period τ . We study the effect of the length of sampling period τ in data histogram construction. Specifically, we vary τ among 1, 3, 5, 7, 9, 11, and 13 days. Due to the space limitation, we report on the effect of the length of sampling period τ on dataset **PEMS04-Evolve** only. The results on **PEMS03-Evolve** show similar trends. Fig. 15 shows the results w.r.t. *MAE*, *RMSE*, *MAPE*, and *runtime*. The proposed framework achieves the best performance when $\tau = 7$ days, which is the default value. When $\tau < 7$ days, the constructed histogram does not have complete observations of traffic flow patterns. Many temporal patterns, such as the correlation between weekdays and weekends, are excluded from the observations. In this case, EMD might not accurately measure the consistency of existing nodes. When $\tau > 7$ days, the results are insignificantly improved while the buffer size $|\mathcal{B}_c|$ has to store a larger amount of historical data, which leads to increased computational and storage costs.

5 RELATED WORK

Traffic Forecasting. Capturing spatio-temporal dynamics is one of the most essential aspects of traffic forecasting models. Thus, a variety of neural network based methods that build on GNNs [14] and TCNs [13] have been proposed that aim to capture complex spatio-temporal dynamics, thereby achieving competitive forecasting performance. Yu et al. [59] propose a traffic forecasting framework that combines GCNs and 1DCNNs in a “sandwich” architecture. Li et al. [35] propose a traffic forecasting model using a sequence-to-sequence architecture that combines GCNs and Recurrent Neural Networks (RNNs). Wu et al. [54] combines GCNs with WaveNet [50], which is an advanced 1DCNN. Attention mechanisms are also used for modeling spatial and temporal information. Zheng et al. [60] propose a framework that employs spatial attentions to model the correlations among multiple time series in an RN and employs temporal attentions to model the importance of each time step in a time series. Razvan et al. [14] propose an attention-based framework that encompasses location-specific and time-varying model parameters to better capture complex spatio-temporodynamics. Jiang et al. [27] model the time delay in spatial information propagation and use a masking mechanism to model both short-range and long-range spatial dependencies. Motivated by meta-learning, MetaStore [37] and MetaST [57] enable transfer of knowledge from a data-abundant source city to a data-limited target city. Lanza et al. [33] propose a framework that combines federated learning and continual learning to sequentially train a global traffic forecasting model from the traffic signals of different local nodes. However, existing studies only work on static time series and fixed-topology RNs. The most relevant study to ours is **TrafficStream** [8] that also employs a subset of nodes of RNs to efficiently capture traffic signal. However, **TrafficStream** can contend with only the expansion of RNs and cannot work on evolved RNs, where RNs can expand, shrink, or

undergo topological updates. Further, **TrafficStream** does not work well on small-scale data that is typically available for newly expanded regions. In contrast, TEAM contends with these aspects and works on evolved RNs. In particular, TEAM works well on small-scale data due to its hybrid architecture. To the best of our knowledge, TEAM is the first proposal to contend with incremental time series and evolving RNs.

Dynamic Graph Embedding. Approaches exist that model dynamic graphs by capturing the progression of topology changing over time [28]. DynGEM [18] exploits graph-autoencoders for incrementally updating node embedding by initializing them based on the previous step. DyRep [49] employs point processes to dynamically model edge occurrences between changing nodes. Dynamic-Triad [61] focuses on the specific structure of triads to model how closed triads (three interconnected vertices) are formed from open triads (three vertices not interconnected). HTNE [62] captures dynamics by employing the Hawkes process and an attention mechanism to assess the impact of historical neighbors on the current neighbors of a node. EvolveGCN [44] adopts GCNs to generate node embeddings for each snapshot and utilizes RNNs to train the GCNs. These approaches face the notable scalability issue that they must fully train the models on the entire graph at every timestep. To the best of our knowledge, our proposed framework is the first study that considers efficiency in model training.

Rehearsal-based Continual Learning. Rehearsal-based methods avoid catastrophic forgetting in continual learning by replaying a subset of old representative samples stored in a size-constrained memory buffer. Numerous studies propose algorithms to choose the most representative samples for the buffer [41]. Chaudhry et al. [7] propose reservoir sampling that guarantees that each input sample has the same probability of entering the buffer. Lopez et al. [38] propose a ring buffer that allocates an equal-sized buffer to each class. Aljundi et al. [1] propose a gradient-based sampling algorithm to reduce overfitting of the reservoir and the ring algorithms by maximizing the diversity of samples in the buffer. However, rehearsal-based continual learning algorithms are only applied to perform classification tasks. To the best of our knowledge, our study is the first to adapt rehearsal-based continual learning for traffic forecasting, a regression problem.

6 CONCLUSION AND FUTURE WORK

We present Topological Evolution-aware Framework (TEAM), a framework for solving traffic forecasting in evolving RNs. For the core of the framework, we propose a spatio-temporal model with a hybrid architecture, namely Convolution Attention for Spatio-Temporal (CAST), that combines convolution and attention to adapt better to incremental time series. We propose a continual learning module based on the rehearsal method and integrate the module into the framework. The continual module works as a buffer to store limited time series subsequences from the most stable and the most unstable nodes. The model is then trained on the data of newly added nodes and the data in the buffer. Experimental studies show that the framework is capable of outperforming strong baselines and state-of-the-art methods.

In future research, it is of interest to study traffic forecasting with only a limited amount of training data [17, 39]. It is also of interest to attempt to further improve the continual learning module, e.g., by identifying a better strategy for selecting representative time series subsequences [58], by capturing temporal information better [10, 48, 53], or by modeling the continual learning using a generative model [2]. Further, it is of interest to consider distributed model training [36].

ACKNOWLEDGMENTS

We thank Khanh-Toan Nguyen and Thin Nguyen from A²I², Deakin University, Australia for fruitful discussions and technical help.

REFERENCES

- [1] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. 2019. Gradient-based Sample Selection for Online Continual Learning. In *Conference on Neural Information Processing Systems (NeurIPS)*. 11816–11825.
- [2] Ali Ayub and Alan R. Wagner. 2021. EEC: Learning to Encode and Regenerate Images for Continual Learning. In *International Conference on Learning Representations (ICLR)*. 1–16.
- [3] Ulrik Brandes and Thomas Erlebach (Eds.). 2005. *Network Analysis: Methodological Foundations [outcome of a Dagstuhl seminar, 13–16 April 2004]*. Lecture Notes in Computer Science, Vol. 3418. Springer.
- [4] David Campos, Tung Kieu, Chenjuan Guo, Feiteng Huang, Kai Zheng, Bin Yang, and Christian S. Jensen. 2021. Unsupervised Time Series Outlier Detection with Diversity-Driven Convolutional Ensembles. *Proc. VLDB Endow.* 15, 3 (2021), 611–623.
- [5] David Campos, Bin Yang, Tung Kieu, Miao Zhang, Chenjuan Guo, and Christian S. Jensen. 2024. QCore: Data-Efficient, On-Device Continual Calibration for Quantized Models. *Proc. VLDB Endow.* 17, 11 (2024), 2708–2721.
- [6] Manoel Castro-Neto, Younseon Jeong, Myong Kee Jeong, and Lee D. Han. 2009. Online-SVR for short-term traffic flow prediction under typical and atypical traffic conditions. *Expert Syst. Appl.* 36, 3 (2009), 6164–6173.
- [7] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. 2019. Efficient Lifelong Learning with A-GEM. In *International Conference on Learning Representations (ICLR)*. 1–15.
- [8] Xu Chen, Junshan Wang, and Kunqing Xie. 2021. TrafficStream: A Streaming Traffic Flow Forecasting Framework Based on Graph Neural Networks and Continual Learning. In *International Joint Conferences on Artificial Intelligence (IJCAI)*. 3620–3626.
- [9] Xinyang Chen, Sinan Wang, Bo Fu, Mingsheng Long, and Jianmin Wang. 2019. Catastrophic Forgetting Meets Negative Transfer: Batch Spectral Shrinkage for Safe Transfer Learning. In *Conference on Neural Information Processing Systems (NeurIPS)*. 1906–1916.
- [10] Yunyao Cheng, Chenjuan Guo, Bin Yang, Haomin Yu, Kai Zhao, and Christian S. Jensen. 2024. A Memory Guided Transformer for Time Series Forecasting. *Proc. VLDB Endow.* 18 (2024).
- [11] Junyoung Chung, Çağlar Gülcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv preprint* (2014). arXiv:1412.3555
- [12] Razvan-Gabriel Cirstea, Chenjuan Guo, Bin Yang, Tung Kieu, Xuanyi Dong, and Shirui Pan. 2022. Triformer: Triangular, Variable-Specific Attentions for Long Sequence Multivariate Time Series Forecasting. In *International Joint Conference on Artificial Intelligence (IJCAI)*. 1994–2001.
- [13] Razvan-Gabriel Cirstea, Tung Kieu, Chenjuan Guo, Bin Yang, and Sinnor Jialin Pan. 2021. EnhanceNet: Plugin Neural Networks for Enhancing Correlated Time Series Forecasting. In *IEEE International Conference on Data Engineering (ICDE)*. 1739–1750.
- [14] Razvan-Gabriel Cirstea, Bin Yang, Chenjuan Guo, Tung Kieu, and Shirui Pan. 2022. Towards Spatio-Temporal Aware Traffic Time Series Forecasting. In *IEEE International Conference on Data Engineering (ICDE)*. 2900–2913.
- [15] Zihang Dai, Hanxiao Liu, Quoc V. Le, and Mingxing Tan. 2021. CoAtNet: Marrying Convolution and Attention for All Data Sizes. In *Conference on Neural Information Processing Systems (NeurIPS)*. 3965–3977.
- [16] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *Conference on Neural Information Processing Systems (NeurIPS)*. 3837–3845.
- [17] Begüm Demir, Francesco Bovolo, and Lorenzo Bruzzone. 2013. Classification of Time Series of Multispectral Images With Limited Training Data. *IEEE Trans. Image Process.* 22, 8 (2013), 3219–3233.
- [18] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. 2017. DynGEM: Deep Embedding Method for Dynamic Graphs. In *International Joint Conferences on Artificial Intelligence (IJCAI) Workshop on Representation Learning for Graphs*. 22–31.
- [19] Chenjuan Guo, Ronghui Xu, Bin Yang, Ye Yuan, Tung Kieu, Yan Zhao, and Christian S. Jensen. 2024. Efficient stochastic routing in path-centric uncertain road networks. *Proc. VLDB Endow.* 17, 11 (2024), 2893–2905.
- [20] Lan-Zhe Guo, Zhi Zhou, and Yu-Feng Li. 2020. RECORD: Resource Constrained Semi-Supervised Learning under Distribution Shift. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (SIGKDD)*. 1636–1644.
- [21] Shengnan Guo, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. 2019. Attention Based Spatial-Temporal Graph Convolutional Networks for Traffic Flow Forecasting. In *AAAI Conference on Artificial Intelligence (AAAI)*. 922–929.
- [22] James D. Hamilton. 1994. *Time Series Analysis*. Vol. 2. Princeton University Press.
- [23] Mingguo He, Zhewei Wei, and Ji-Rong Wen. 2022. Convolutional Neural Networks on Graphs with Chebyshev Approximation, Revisited. *CoRR* abs/2202.03580 (2022).
- [24] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (1997), 1735–1780.
- [25] Peter J. Huber. 1992. *Robust Estimation of a Location Parameter*. Springer New York.
- [26] Robin John Hyndman and George Athanasopoulos. 2018. *Forecasting: Principles and Practice* (2nd ed.). OTexts, Australia.
- [27] Jiawei Jiang, Chengkai Han, Wayne Xin Zhao, and Jingyuan Wang. 2023. PDFFormer: Propagation Delay-Aware Dynamic Long-Range Transformer for Traffic Flow Prediction. In *AAAI Conference on Artificial Intelligence (AAAI)*. 4365–4373.
- [28] Seyed Mehran Kazemi, Rishabh Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. 2020. Representation Learning for Dynamic Graphs: A Survey. *J. Mach. Learn. Res.* 21 (2020), 70:1–70:73.
- [29] Tung Kieu, Bin Yang, Chenjuan Guo, and Christian S. Jensen. 2018. Distinguishing Trajectories from Different Drivers using Incompletely Labeled Trajectories. In *International Conference on Information and Knowledge Management (CIKM)*. 863–872.
- [30] Tung Kieu, Bin Yang, Chenjuan Guo, and Christian S. Jensen. 2019. Outlier Detection for Time Series with Recurrent Autoencoder Ensembles. In *International Joint Conferences on Artificial Intelligence (IJCAI)*. 2725–2732.
- [31] Tung Kieu, Bin Yang, and Christian S. Jensen. 2018. Outlier Detection for Multidimensional Time Series Using Deep Neural Networks. In *IEEE International Conference on Mobile Data Management (MDM)*. 125–134.
- [32] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR)*. 1–14.
- [33] Chiara Lanza, Eduard Angelats, Marco Miozzo, and Paolo Dini. 2023. Urban Traffic Forecasting using Federated and Continual Learning. In *Conference on Cloud and Internet of Things (CIoT)*. 1–8.
- [34] Jure Leskovec and Christos Faloutsos. 2006. Sampling from Large Graphs. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (SIGKDD)*. 631–636.
- [35] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. 2018. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. In *International Conference on Learning Representations (ICLR)*. 1–16.
- [36] Zhiyu Liang and Hongzhi Wang. 2022. FedTSC: A Secure Federated Learning System for Interpretable Time Series Classification. *Proc. VLDB Endow.* 15, 12 (2022), 3686–3689.
- [37] Yan Liu, Bin Guo, Daqing Zhang, Djamel Zeghlache, Jingmin Chen, Sizhe Zhang, Dan Zhou, Xinlei Shi, and Zhiwei Yu. 2021. MetaStore: A Task-adaptive Meta-learning Model for Optimal Store Placement with Multi-city Knowledge Transfer. *ACM Trans. Intell. Syst. Technol.* 12, 3 (2021), 28:1–28:23.
- [38] David Lopez-Paz and Marc'Aurelio Ranzato. 2017. Gradient Episodic Memory for Continual Learning. In *Conference on Neural Information Processing Systems (NeurIPS)*. 6467–6476.
- [39] Hao Miao, Ziqiao Liu, Yan Zhao, Chenjuan Guo, Bin Yang, Kai Zheng, and Christian S. Jensen. 2024. Less is More: Efficient Time Series Dataset Condensation via Two-fold Modal Matching. *Proc. VLDB Endow.* 18 (2024).
- [40] Vinod Nair and Geoffrey E. Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *International Conference on Machine Learning (ICML)*. 807–814.
- [41] Toan Nguyen, Duc Kieu, Bao Duong, Tung Kieu, Kien Do, Thin Nguyen, and Bac Le. 2024. Class-incremental Learning with Causal Relational Replay. *Expert Syst. Appl.* 250 (2024), 123901.
- [42] Yuki Ono, Eduard Trulls, Pascal Fua, and Kwang Moo Yi. 2018. LF-Net: Learning Local Features from Images. In *Conference on Neural Information Processing Systems (NeurIPS)*. 6237–6247.
- [43] Boris N. Oreshkin, Dmitrii Carpow, Nicolas Chapados, and Yoshua Bengio. 2020. N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. In *International Conference on Learning Representations (ICLR)*. 1–31.
- [44] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao B. Schardl, and Charles E. Leiserson. 2020.

- EvolveGCN: Evolving Graph Convolutional Networks for Dynamic Graphs. In *AAAI Conference on Artificial Intelligence (AAAI)*. 5363–5370.
- [45] Zehui Shao, Zhao Zhang, Wei Wei, Fei Wang, Yongjun Xu, Xin Cao, and Christian S. Jensen. 2022. Decoupled Dynamic Spatial-Temporal Graph Neural Network for Traffic Forecasting. *Proc. VLDB Endow.* 15, 11 (2022), 2733–2746.
- [46] Alexander Soen and Ke Sun. 2021. On the Variance of the Fisher Information for Deep Learning. In *Conference on Neural Information Processing Systems (NeurIPS)*. 5708–5719.
- [47] Chao Song, Youfang Lin, Shengnan Guo, and Huaiyu Wan. 2020. Spatial-Temporal Synchronous Graph Convolutional Networks: A New Framework for Spatial-Temporal Network Data Forecasting. In *AAAI Conference on Artificial Intelligence (AAAI)*. 914–921.
- [48] Jianwei Tang, Jiangxin Sun, Xiaotong Lin, Lifang Zhang, Wei-Shi Zheng, and Jian-Fang Hu. 2023. Temporal Continual Learning with Prior Compensation for Human Motion Prediction. In *Conference on Neural Information Processing Systems (NeurIPS)*. 1–13.
- [49] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2019. DyRep: Learning Representations over Dynamic Graphs. In *International Conference on Learning Representations (ICLR)*. 1–25.
- [50] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. 2016. WaveNet: A Generative Model for Raw Audio. *CoRR* abs/1609.03499 (2016).
- [51] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations (ICLR)*. 1–12.
- [52] Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. 2021. CvT: Introducing Convolutions to Vision Transformers. In *International Conference on Computer Vision (ICCV)*. 22–31.
- [53] Xinle Wu, Dalin Zhang, Miao Zhang, Chenjuan Guo, Bin Yang, and Christian S. Jensen. 2023. AutoCTS+: Joint Neural Architecture and Hyperparameter Search for Correlated Time Series Forecasting. *Proc. ACM Manag. Data* 1, 1 (2023), 97:1–97:26.
- [54] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. 2019. Graph WaveNet for Deep Spatial-Temporal Graph Modeling. In *International Joint Conferences on Artificial Intelligence (IJCAI)*. 1907–1913.
- [55] Cuie Yang, Yiu-Ming Cheung, Jinliang Ding, and Kay Chen Tan. 2022. Concept Drift-Tolerant Transfer Learning in Dynamic Environments. *IEEE Trans. Neural Networks Learn. Syst.* 33, 8 (2022), 3857–3871.
- [56] Sean Bin Yang, Jili Hu, Chenjuan Guo, Bin Yang, and Christian S. Jensen. 2023. LightPath: Lightweight and Scalable Path Representation Learning. In *KDD*. ACM, 2999–3010.
- [57] Huaxiu Yao, Yiding Liu, Ying Wei, Xianfeng Tang, and Zhenhui Li. 2019. Learning from Multiple Cities: A Meta-Learning Approach for Spatial-Temporal Prediction. In *International World Wide Web Conference (WWW)*. 2181–2191.
- [58] Jaehong Yoon, Divyam Madaan, Eunho Yang, and Sung Ju Hwang. 2022. Online Coreset Selection for Rehearsal-based Continual Learning. In *International Conference on Learning Representations (ICLR)*. 1–15.
- [59] Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2018. Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting. In *International Joint Conferences on Artificial Intelligence (IJCAI)*. 3634–3640.
- [60] Chuapan Zheng, Xiaoliang Fan, Cheng Wang, and Jianzhong Qi. 2020. GMAN: A Graph Multi-Attention Network for Traffic Prediction. In *AAAI Conference on Artificial Intelligence (AAAI)*. 1234–1241.
- [61] Le-kui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueteng Zhuang. 2018. Dynamic Network Embedding by Modeling Triadic Closure Process. In *AAAI Conference on Artificial Intelligence (AAAI)*. 571–578.
- [62] Yuan Zuo, Guannan Liu, Hao Lin, Jia Guo, Xiaoqian Hu, and Junjie Wu. 2018. Embedding Temporal Network via Neighborhood Formation. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (SIGKDD)*. 2857–2866.