DOI: 10.1111/tgis.12644

RESEARCH ARTICLE

Transactions in GIS WILEY

# Traffic transformer: Capturing the continuity and periodicity of time series for traffic forecasting

Ling Cai[1] | Krzysztof Janowicz[1] | Gengchen Mai[1] | Bo Yan[2] | Rui Zhu[1]

[1]Department of Geography, University of California, Santa Barbara, CA, USA

[2]LinkedIn, Mountain View, CA, USA

**Correspondence**
Ling Cai, Department of Geography, University of California, Santa Barbara, CA 93106-9010, USA.
Email: lingcai@ucsb.edu

## Abstract

Traffic forecasting is a challenging problem due to the complexity of jointly modeling spatio-temporal dependencies at different scales. Recently, several hybrid deep learning models have been developed to capture such dependencies. These approaches typically utilize convolutional neural networks or graph neural networks (GNNs) to model spatial dependency and leverage recurrent neural networks (RNNs) to learn temporal dependency. However, RNNs are only able to capture sequential information in the time series, while being incapable of modeling their periodicity (e.g., weekly patterns). Moreover, RNNs are difficult to parallelize, making training and prediction less efficient. In this work we propose a novel deep learning architecture called *Traffic Transformer* to capture the continuity and periodicity of time series and to model spatial dependency. Our work takes inspiration from Google's Transformer framework for machine translation. We conduct extensive experiments on two real-world traffic data sets, and the results demonstrate that our model outperforms baseline models by a substantial margin.

## 1 | INTRODUCTION

Traffic forecasting is concerned with estimating future traffic conditions (such as the density of vehicles and their speed) to enable the prediction of future events (such as congestion or travel duration) by analyzing historical traffic conditions and patterns. Highly accurate forecasts provide guidance to decision-makers, provide safety and convenience for citizens, and reduce environmental impacts.

Traffic forecasting, however, is challenging due to the complexity of modeling spatio-temporal dependencies of traffic conditions at varying scales (Davis, Raina, & Jagannathan, 2019; Wu, Tan, Qin, Ran, & Jiang, 2018). For instance, the traffic flow on a road is influenced by both its historical traffic conditions and the conditions of upstream roads. Due to the increasing availability of massive traffic data, high-performance computing, and novel deep learning models, recent work has pushed the envelope on learning spatio-temporal dependency models for accurate traffic forecasting (Cui, Ke, & Wang, 2018; Jin, Lin, Wu, & Wan, 2018; Wu et al., 2018; Yao, Tang, Wei, Zheng, & Li, 2019; Yu, Yin, & Zhu, 2017).

Models based on recurrent neural networks (RNNs), such as Gated Recurrent Unit (GRU) and Long Short Term Memory (LSTM), can be used effectively to capture temporal dependencies (Cui et al., 2018; Jin et al., 2018). For example, Shi et al. (2015) proposed a convolutional LSTM model for traffic forecasting, in which the traffic flow at each time-step was recursively fed into an LSTM architecture. Similarly, Li, Yu, Shahabi, and Liu (2017) proposed a convolutional RNN model where graph diffusion convolutional operators were used to model spatial dependencies and GRU was employed instead of LSTM to capture temporal dependencies.

Although these RNN-based models can capture temporal sequential dependency, RNNs have several inherent deficiencies. First, they struggle to preserve very long-term sequential information, which leads to the loss of long-term temporal dependency in time series during the forward path (Khandelwal, He, Qi, & Jurafsky, 2018). Second, RNNs are unable to capture the periodicity of time series, since they treat different time-steps equally in the time series. This is an important shortcoming as time series usually convey periodic patterns, such as hourly, daily, weekly, and seasonally (Guo, Lin, Feng, Song, & Wan, 2019; Yao et al., 2019). Third, RNNs are difficult to parallelize, making the training and prediction process less efficient.

Recently, researchers have introduced the Transformer architecture to replace RNNs for machine translation (Vaswani et al., 2017). It replaces convolutional neural networks (CNNs) and RNNs and is solely built on attention mechanisms to model sequential data. Hence, Transformer does not require sequential data to be fed recursively. This makes the architecture computationally more efficient than RNNs. More importantly, so as to preserve the order of elements in a sequence (e.g., the order of words in a sentence) when modeling sequences, Transformer introduces a position encoding strategy. It encodes positions of elements (e.g., words) in the sequence (e.g., a sentence) by first indexing them by their positions and then passing the indexes through a series of sinusoidal functions. Transformer and its variants have achieved significant success in natural language processing, including machine translation and text generation (Dai et al., 2019; Devlin, Chang, Lee, & Toutanova, 2018; Radford, Narasimhan, Salimans, & Sutskever, 2018).

Interestingly, machine translation and traffic prediction share some structural similarities. In machine translation (Wu et al., 2016), the aim is to translate a source sentence written in one language into a target sentence in another language by using a sequence-to-sequence learning framework (Sutskever, Vinyals, & Le, 2014), where the source and target sequences both consist of tokens. Traffic prediction can be formulated in a similar way. More specifically, the task is to utilize data about historical traffic conditions in such a way that they become indicative of future conditions, where the source sequence consists of a series of traffic data (e.g., traffic volume, speed) in the past and the target sequence is composed of a series of traffic conditions at future time-steps. Put differently, each time-step in the source and target traffic sequences can be analogized to the position index of each word in the input and output sentences in the machine translation task.

Analogies are partial by definition. Hence, Transformer cannot be directly applied to traffic forecasting for the following reasons. First, the position encoding strategy is inapplicable. The semantics of the source and target sequences in machine translation and traffic forecasting are different. In machine translation, the source and target sequences represent two sentences with the same meaning in different languages; thus, the corresponding words in the two sequences should share the same position index. In contrast, in traffic forecasting, the source–target sequence is consecutive; hence, there is no correspondence between elements in the source and target sequences. Instead, traffic forecasting takes into account the continuity of time series when indexing the traffic by their time-steps. Additionally, traffic data are also characterized by several other properties of time, such as

periodicity. For instance, the traffic conditions on one road on Wednesday at 3:00 p.m. are similar to the traffic conditions on Thursday at the same time. The periodic characteristics of traffic data should also be considered when adapting Transformer to this domain. Consequently, this calls for new ways of encoding temporal features in the Transformer architecture. Second, Transformer is only able to handle the sequential dependency between the source and target sequences in machine translation, while spatial (network information) and temporal (sequential information) dependencies in traffic data are prominent (Cui, Henrickson, Ke, & Wang, 2019; Ma et al., 2017). Hence, we need to enable Transformer to handle spatial and temporal dependencies coherently.

To solve these problems, we propose to design different strategies for encoding temporal information so that both the continuity and periodicity of traffic data can be preserved, and extend Transformer to modeling temporal and spatial dependencies jointly with the help of graph convolutional networks (GCNs). The main contributions of our research are as follows.

1. We design four novel position encoding strategies to encode the continuity and periodicity of time series to facilitate the modeling of temporal dependencies in traffic data. In total, we propose seven temporal encoding methods by combining different strategies.
2. We introduce a hybrid encoder–decoder architecture, called *Traffic Transformer*, to coherently model spatial and temporal dependencies of traffic data in an end-to-end training manner, where Transformer is leveraged to model temporal dependencies and GCNs contribute to the modeling of spatial dependencies.
3. Experimental results on two real-world benchmark data sets show the performance of our model compared to state-of-the-art methods, demonstrating the effectiveness of our temporal encoding methods and the hybrid architecture.

The remainder of this article is structured as follows. Section 2 reviews existing work on traffic forecasting. Section 3 defines the traffic forecasting task and introduces Transformer. Section 4 presents different strategies for encoding temporal characteristics and the proposed architecture for traffic forecasting. Section 5 explains our experiments and presents the results. Finally, Section 6 concludes our work and points to directions for future research.

## 2 | RELATED WORK

Driven by big data machine learning models for traffic forecasting have attracted considerable attention from both academia and industry for several years (Dai et al., 2019; Liu, Zheng, Chawla, Yuan, & Xing, 2011; Lv, Duan, Kang, Li, & Wang, 2014; Yao et al., 2019; Zhao et al., 2019). As far as deep learning is concerned, Huang, Song, Hong, and Xie (2014) were among the first to apply deep learning models to forecasting traffic, by designing a deep belief network for unsupervised feature learning and then passing these features through a regression layer for traffic forecasting. Since then, most deep learning models for traffic forecasting have been built by utilizing RNNs due to their ability to memorize temporal dependencies in time series via self-circulation. Fu, Zhang, and Li (2016) compared different RNN models, namely LSTM and GRU, finding that GRU achieved better performance than LSTM in forecasting traffic. However, these methods are limited to capturing forward temporal dependencies. In contrast, Cui et al. (2018) proposed a deeply stacked bidirectional and unidirectional LSTM architecture which is able to capture both forward and backward dependencies in time series.

While the aforementioned models account for temporal dependencies in traffic data, spatial correlations have often been neglected. To fill the gap, Lv et al. (2014) proposed a novel deep learning architecture to inherently consider temporal and spatial dependencies, where autoencoders were first introduced to serve as the building block to learn latent features. Zhang, Zheng, Qi, Li, and Yi (2016) designed a deep learning model which takes into account both temporal aspects (temporal closeness, periods, and trends of crowd traffic) and spatial

proximity. More recently, CNNs and graph neural networks (GNNs) have become the most widely used models in detecting patterns thanks to their progress in capturing spatial and topological dependencies in images, videos, and graphs (Ke, Zheng, Yang, & Chen, 2017; Li et al., 2017; Wu et al., 2018; Yu et al., 2017). In addition to a GRU for capturing temporal features, Cao et al. (2017) converted network-wide traffic matrices into images, after which a CNN was imposed to learn global spatial interactions in the converted images. Wu et al. (2018) proposed a hybrid deep learning framework by marrying the CNN with an LSTM architecture, where a one-dimensional CNN was utilized to model spatial dependencies and two LSTMs were exploited to learn temporal patterns. Considering that a graph is a more appropriate abstraction of a road network, Li et al. (2017) suggested replacing CNNs with GCNs in order to extract spatial dependencies, and proposed a new model, called DCRNN, in which spatial dependencies are modeled as a diffusion process. Similarly, Cui et al. (2019) defined another graph convolutional operator, incorporating the adjacency matrix and a free-flow reachable matrix to capture localized spatial features. More generally, the future seems to lie in combining CNNs/GCNs with RNNs to extract both spatial and temporal dependencies for traffic forecasting. However, one limitation of such architectures stems from RNNs themselves, which are well known for their high computational cost during the training process.

To circumvent the inherent deficiencies of RNNs, researchers have started to investigate non-recurrent models. For example, Yu, Yin, and Zhu (2018) proposed a universal framework that consists entirely of spatio-temporal convolutional blocks. Experiments showed that this model yielded better results than the model proposed by Li et al. (2017). Guo et al. (2019) designed a spatio-temporal attention mechanism as well as a spatio-temporal convolutional module to capture spatio-temporal dependencies of traffic data. However, this approach relied solely on the temporal attention mechanism to assign different importance to traffic in the past, and by doing so it ignored the fact that the most recent traffic condition should have a greater influence on predicting current traffic.

Recently, Transformer (Vaswani et al., 2017) has been developed as a new architecture in deep learning, which employs attention mechanisms along with a position encoding strategy for sequence modeling. In light of this, several attempts have been made to tailor Transformer to time series forecasting (Li et al., 2019; Li & Moura, 2019; Lim, Arik, Loeff, & Pfister, 2019). For example, Li et al. (2019) argued that the basic Transformer architecture is not sensitive to local contexts, and suggested adding a convolutional self-attention layer to improve it. Although this work has been demonstrated to be effective in capturing long-term temporal dependencies, it is insufficient for network-wide traffic forecasting. First, it ignored spatial dependencies on a road network and thus failed to model spatio-temporal correlations. Second, it captured periodical patterns at the cost of feeding very long sequences (e.g., 768) into models, which may be impossible for network-wide traffic forecasting. The amount of traffic data in a network is far larger than that at a single spot as there are hundreds of sensors on a road network reporting the traffic frequently. Li and Moura (2019) have also drawn on the idea of Transformer to cope with time series problems and proposed a novel method, called Forecaster, to address the problem of forecasting taxi ride-hailing demand. However, their focus is on how to better model spatial dependencies by using sparse linear layers guided by spatial dependency graphs. The work most similar to ours is Xu et al. (2020), which also focused on jointly modeling spatial and temporal dependencies by making full use of Transformer to solve traffic prediction task. In their paper, they proposed position embeddings as learnable parameters that were learned automatically at training. Thus, the model itself is responsible for capturing the temporal dependencies in time series, and human knowledge of temporal patterns in time series (e.g., the continuity/periodicity of time series) is ignored. By contrast, in this article, we study how to design different temporal encoding strategies by explicitly modeling temporal patterns reflected in the time series. By doing so, prior human knowledge on temporal patterns is regarded as inductive bias to guide the model to capture the temporal dependencies. To summarize, in our article, we focus on explicitly modeling spatial and temporal dependencies all together: the spatial correlations of the traffic in a network as well as the continuity and periodicity of time series.

## 3 | PRELIMINARIES

### 3.1 | The network-wide traffic forecasting task

The aim of traffic forecasting is to predict future traffic, given a sequence of historical traffic observations (here speed, density, and volume) that are detected by sensors on a road network. Such a sensor network deployed to monitor roads is usually represented as a weighted directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{W})$, where $\mathcal{V}$ is a set of sensors with $|\mathcal{V}| = N$, $\mathcal{E}$ is a set of edges connecting sensors, and $\mathbf{W} \in \mathbb{R}^{N \times N}$ is the adjacency matrix storing the distance between sensors in the network. $\mathbf{X}^t \in \mathbb{R}^{N \times P}$ denotes the feature matrix of the graph that is observed at time $t$, where $P$ is the number of features. The prediction problem can then be formalized as learning a mapping function $F$ from $M$ previously observed feature matrices to $H$ future feature matrices on the premise of a network $\mathcal{G}$:

$$\mathcal{X}_{t+1}^{t+H} = F(\mathcal{G}; \mathcal{X}_{t-(M-1)}^{t}) \tag{1}$$

where $\mathcal{X}_i^{i+n}$ denotes an array of feature matrices from time stamp $i$ to $i+n$: $[\mathbf{X}^i, \mathbf{X}^{i+1}, \ldots, \mathbf{X}^{i+n}]$.

### 3.2 | Transformer in machine translation

Unlike RNNs, Transformer belongs to the family of non-RNNs. It is built entirely upon attention mechanisms, which makes it possible to access any part of a sequence regardless of its distance to the target (Li et al., 2019; Vaswani et al., 2017).

In essence, Transformer is organized in an encoder–decoder manner, in which identical encoder modules are stacked at the bottom of stacked decoder modules. Each encoder module is composed of a multi-head self-attention layer and a position-wise feedforward layer, while each decoder module has one more layer (i.e., encoder–decoder attention layer), which is inserted between the self-attention and feedforward layers to bridge the encoder and decoder parts.

The aim of multi-head attention layers is to attach different importance of words/tokens to each other in a sequence from multiple aspects (heads). Then the outputs of those different heads are concatenated and passed through a linear transformation to aggregate all the information. As the attention mechanisms behind the self-attention and encoder–decoder attention layers are the same, we take the self-attention layer as an example. It operates on a sequence of tokens $x = (x_1, x_2, \ldots, x_n)$, each of which is initialized by a random vector, and is updated by using a weighted sum of any other word after being passed through a linear transformation. The weights, called attention scores, are assigned by their similarities.

Take the update of $\mathbf{x}_i$ as an example:

$$\mathbf{y}_i = \sum_{j=1}^{n} a_{ij}(\mathbf{x}_j \mathbf{W}_V) \tag{2}$$

where $\mathbf{y}_i$ is the updated $\mathbf{x}_i$, and $a_{ij}$ is the attention score, measuring the similarity between $\mathbf{x}_i$ and $\mathbf{x}_j$, calculated as,

$$a_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{n} \exp(e_{ik})} \tag{3}$$

where $e_{ij}$ measures the compatibility of two linearly transformed $\mathbf{x}_i$ and $\mathbf{x}_j$, calculated by using the scaled dot product,

$$e_{ij} = \frac{(\mathbf{x}_i \mathbf{W}_Q)(\mathbf{x}_j \mathbf{W}_K)^T}{\sqrt{h}} \tag{4}$$

where $h$ is the dimension of the output. Note that $\mathbf{W}_V$, $\mathbf{W}_Q$, $\mathbf{W}_K$ are three linear transformation matrices to strengthen the expressiveness of Transformer.

More importantly, since Transformer does not involve any convolutional and recurrent module, position embeddings are designed to preserve the sequential order in a sequence. The positions of words in a sentence are first indexed starting from 0 to the length of the sentence and then position indexes are fed into the following expressions to gain position embedding for each word:

$$\begin{aligned} pos\_embedding(pos, 2i) &= \sin\left(pos/10{,}000^{\frac{2i}{h}}\right) \\ pos\_embedding(pos, 2i+1) &= \cos\left(pos/10{,}000^{\frac{2i}{h}}\right) \end{aligned} \tag{5}$$

where $pos$ is the position index of a word in a sequence, $i$ is the $i$th dimension of the position embedding, and $h$ is the dimension of hidden layers. We denote the position embedding of a word at $pos$ by $pos\_embedding_{pos} \in \mathbb{R}^h$. The corresponding words in the source and target sentences share the same position embedding. The encoding strategy guarantees that the position embedding at $pos+k$ is a linear function of that at $pos$. Then an elementwise additive operator is imposed on the position embedding and the initialized vector of their corresponding word (i.e., the feature embedding of a word). The resulting embedding is subsequently fed into encoder/decoder modules.

Although this encoding strategy is useful for machine translation, it is inappropriate for traffic forecasting as temporal features such as continuity and periodicity have to be carefully incorporated.

## 4 | PROPOSED ARCHITECTURE

In this section we introduce four strategies for encoding temporal information, more specifically the *continuity* and *periodicity* of time series. Next, we introduce two graph convolutional filters which can help capture spatial dependencies. Finally, we illustrate our proposed hybrid deep learning architecture.

### 4.1 | Transformer for capturing temporal dependencies

For ease of expression, we assume the sampling frequency of the time series is $s$ times per day. We use $[t - (M-1), t - (M - 2), \ldots, t]$ and $[t + 1, t + 2, \ldots, t + H]$ to denote the time-steps of the source sequence (from the past) and of the target sequence (in the future).

#### 4.1.1 | Continuity of time series

*Relative position encoding*

The aim of this strategy is to encode relative continuity, which means that we care about the continuity of time in the window of the source–target sequence regardless of the *position* of one time-step in the whole time series under consideration. This can be achieved by indexing the time-step $(t - (M - 1))$ with 0 as the starting position and raising the index position by 1 per time-step. These position indexes are simply passed through Equation (5) to get position embeddings for each time-step. By doing so, the continuity of time within the source–target sequence pair is encoded. All concatenated source–target sequences in the traffic prediction task share the same position embeddings.

*Global position encoding*

Despite the success of relative position encoding in preserving the local continuity of time, it ignores the fact that most time-steps in two consecutive source–target sequence pairs are common. For example, the previous

sequence pair may range from 3:00 to 5:00 p.m. with 5 min as the sampling time interval, while the subsequent sequence pair ranges from 3:05 to 5:05 p.m. So the potential limitation of the relative position encoding strategy is that the same time-step is assigned with a different position embedding depending on its position in a sequence pair. Hence, we additionally propose a global/absolute position encoding strategy so that a time-step occurring in the whole time period under consideration has only one position embedding even if it appears in different sequences. All the time-steps in question are sorted by time first and then are indexed starting from 0. As for the relative position encoding strategy, the position embedding for each time-step is obtained by passing the position index into Equation (5). Hence, we assume that both the local and global continuity of time series are preserved.

### 4.1.2 | Periodicity of time series

Aside from the continuity, time series also convey periodicity (i.e., weekly and daily patterns). There are two potential ways to go about this. One is centered around the position encoding design: the position embedding for each time-step is enriched with periodic patterns. The other is by using different time series segments corresponding to different temporal features.

*Periodic position encoding*

Based on any position encoding strategy in Section 4.1.1, here we enrich position embeddings with periodic patterns. The idea is to design another position encoding strategy to cover weekly-periodic and daily-periodic information, which implies applying the relative/global position encoding strategy to position indexes in terms of weeks and days, respectively. For a single day, the sampling times per day are regarded as the number of positions for daily-periodic position encoding. For instance, if the sampling time interval is 5 min, then there are 60 × 24/5 = 288 sampling times per day, that is, the total number of positions we need to encode for a daily-periodic pattern. The derived position embedding is called *daily-periodic position embedding*. On the other hand, weekly-periodic patterns imply that traffic on the same days of the week is more similar as compared to other days. For instance, usually the traffic pattern on a future Friday should be more similar to those from past Fridays than, for example, Wednesdays. In this case, only seven positions are used that correspond to different days of the week. We call the resulting kind of position embedding *weekly-periodic position embedding*. We impose elementwise addition over relative/global position embedding, daily-periodic position embedding, and weekly-periodic position embedding to gain a hybrid position embedding for each time-step. Note that all embeddings are members of the same vector space in $\mathbb{R}^h$, with $h$ being the number of dimensions.

*Time series segments*

In essence, periodic position encoding employs different flags (e.g., flags to differentiate different time-steps on one day and flags to denote days with distinct week attributes) to explicitly differentiate sequences in three aspects. Although it seems compelling to rely solely on the hybrid position embedding, this may hide other commonalities among sequences. It would make training models more difficult, especially when a training data set is small. Hence, one may enrich the existing time series segment by concatenating two more intercepted time series segments, a daily component and weekly component, along the time axis as the input for modeling as Guo et al. (2019) did and then to encode positions in this hybrid segment jointly.

For the daily-periodic segment, denoted by $\mathcal{X}_{t+1}^{t+H}(D)$, we intercept segments in the same time period as the predicting time period on the last few days, say $d$ days. The segment can be formulated as:

$$\mathcal{X}_{t+1}^{t+H}(D=d)=[\mathcal{X}_{t+1-sd}^{t+H-sd},\mathcal{X}_{t+1-s(d-1)}^{t+H-s(d-1)},\ldots,\mathcal{X}_{t+1-s}^{t+H-s}] \tag{6}$$

The weekly-periodic segment, denoted by $\mathcal{X}_{t+1}^{t+H}(W)$, consists of the time series segment at the same time period as the predicting period on the days with the same week attribute in the past few weeks, say $w$ weeks. This segment can be written as:

$$\mathcal{X}_{t+1}^{t+H}(W=w)=[\mathcal{X}_{t+1-s7w}^{t+H-s7w},\mathcal{X}_{t+1-s7(w-1)}^{t+H-s7(w-1)},\ldots,\mathcal{X}_{t+1-s7}^{t+H-s7}] \tag{7}$$

Finally, the hybrid segment, enriched by the recent, daily-periodic and weekly-periodic information, is composed of $[\mathcal{X}_{t+1}^{t+H}(W),\mathcal{X}_{t+1}^{t+H}(D),\mathcal{X}_{t-(M-1)}^{t}]$ as the input to replace $\mathcal{X}_{t-(M-1)}^{t}$ in Equation (1).

Note that although we concatenate these three segments along the time axis, it is impossible for Transformer to determine the order information as it is completely attention-based. Therefore a position encoding strategy is also required. Since the time periods of a daily-periodic and week-periodic sub-segment on one day are the same as the forecasting period, the time-steps in a daily-periodic/week-periodic sub-segment share the same position embeddings as that of the forecasting segment. Here we do not explicitly take into account the different contributions of daily-periodic/week-periodic segments to the prediction and leave it to the model to determine them.

### 4.1.3 | A summary of encoding methods

In total, there are seven different encoding methods for capturing the continuity and periodicity of time series by combining the aforementioned strategies. A summary with concrete encoding examples can be found in Table 1. To show how to encode positions according to our different temporal encoding strategies, we take a specific example as below. Given the traffic at 8:00–8:55 a.m. on January 01, 2020 (Wednesday), we want to predict the future traffic at 9:00–9:55 a.m. on January 01, 2020 (Wednesday) in the table.

After obtaining these position indexes from different encoding methods, they are used in Equation (5) to derive position embeddings for each time-step in the source–target sequence. Similar to the original Transformer, one option for incorporating position embeddings into time series is by elementwise addition between the traffic features and its position embeddings. Although it is easy to implement, it is hard to interpret such a combination since the vector spaces of the traffic features and the position embeddings would not be the same. We call this approach an *addition-based combination*. However, we suggest another approach here, namely a *similarity-based combination*. We adjust the attention score $a_{ij}$ in Equation (2) by using the similarity between two time-steps in terms of position embeddings. In this way, the similarity between two time-steps serves as a decay factor. That is to say, when two time-steps are adjacent, then their similarity is high. So in the traffic prediction task, we can rewrite Equations (2) and (3) as:

$$\mathbf{Y}^i=\sum_{j=1}^{L}a'_{ij}(\mathbf{X}^jW_V) \tag{8}$$

$$a'_{ij}=\frac{\exp(e'_{ij})}{\sum_{k=1}^{L}\exp(e'_{ik})} \tag{9}$$

In Equation (8), $a'_{ij}$ is the adjusted attention score. Note that in the traffic prediction scenario $\mathbf{X}^i\in\mathbb{R}^{N\times h}$ is the traffic at time $i$ and $L$ is the length of a sequence in question. In Equation (9):

$$e'_{ij}=b_{ij}e_{ij} \tag{10}$$

**TABLE 1** A summary of encoding methods

| Embedding method | Relative position embedding | Global position embedding | Relative and periodic embedding | Global and periodic embedding | Time series segment method |
|---|---|---|---|---|---|
| Encoding strategy | Relative position encoding | Global position encoding | Relative position encoding and periodic position encoding | Global position encoding and periodic position encoding | Time series segments |
| Abbreviation | RPE | GPE | RPPE | GPPE | TSE |
| Captured features | Relative continuity of time series | Global continuity of time series | Relative continuity and periodicity of time series | Global continuity and periodicity of time series | Relative continuity and periodicity of time series |
| Example (position index) | [0, 1, ..., 11] → [12, 13, ..., 23] | [99, 100, ..., 110] → [111, 112, ..., 122] | [0, 1, ..., 11] * [3, 3, ... 3] * [97, 98, ..., 108] → [12, 13, ..., 23] | [99, 100, ..., 110] * [3, 3, ..., 3] * [97, 98, ..., 108] → [111, 112, ..., 122] | [12, 13, ..., 23, 12, 13, ..., 23, 0, 1, ..., 11] → [12, 13, ..., 23] |
| Notes | | Assume among the whole time steps in question, January 01, 2020 Wednesday 8:00 a.m. is indexed as 99 | "*" here means we consider these three kinds of position indexes (relative, weekly, daily) | See previous | Here, $d = w = 1$ |

is the adjusted compatibility of two linearly transformed $\mathbf{X}^i$ and $\mathbf{X}^j$ with the similarity between corresponding temporal features being considered. In Equation (10),

$$b_{ij} = \frac{\exp(d_{ij})}{\sum_{k=1}^{L} \exp(d_{ik})} \tag{11}$$

where

$$d_{ij} = pos\_embedding_i \times (pos\_embedding_j)^T \tag{12}$$

is the similarity between two position embeddings of $\mathbf{X}^i$ and $\mathbf{X}^j$ and $e_{ij}$ is defined in Equation (4).

## 4.2 | Graph convolutional neural networks for capturing spatial dependency

Another important dependency we need to address is spatial dependency. Simply put, we need to account for the fact that the change in traffic on one road is influenced by the traffic on its upstream roads. Since the traffic network is modeled as a graph and the traffic observations (speed, volume, etc.) can be considered as features of that road (at some time), we take advantage of GNNs to perform the convolution operation over graph-structured data to capture topological properties such as adjacency.

In general, a GNN built on spectral graph theory is a generalization of a traditional CNN. In spectral graph theory, a graph is usually represented by its Laplacian matrix. The topological features of a graph can be obtained by analyzing the Laplacian matrix. Formally, graph spectral convolution can be defined as the multiplication of a signal $\mathbf{X} \in \mathbb{R}^N$ by a kernel $g_\theta$, written as:

$$g_\theta *_G \mathbf{X} = \mathbf{U} g_\theta(\Lambda) \mathbf{U}^T \mathbf{X} \tag{13}$$

where $*_G$ is the graph convolutional operator, and $\mathbf{U} \in \mathbb{R}^{N \times N}$ is the matrix of eigenvectors decomposed from the normalized graph Laplacian $\mathbf{L} = \mathbf{I}_N - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}} = \mathbf{U}\Lambda\mathbf{U}^T \in \mathbb{R}^{N \times N}$, where $\mathbf{I}_N$ is an identity matrix, $\mathbf{D} \in \mathbb{R}^{N \times N}$ is the diagonal degree matrix and \boldsymbol\Lambda is a diagonal matrix of its eigenvalues.

Since it is computationally expensive to directly decompose the Laplacian matrix, especially when a graph is large, several approximation approaches have been proposed. Two kinds of graph convolutional operators based on different approximation strategies have been used for traffic forecasting.

The most popular approximation was proposed by Kipf and Welling (2016). Basically they followed the idea of Hammond, Vandergheynst, and Gribonval (2011) that $g_\theta$ can be well approximated by a truncated expansion of Chebyshev polynomials. But they only adopted the first-order polynomials as the graph convolutional filter, since this is more computationally efficient. This setting can be achieved by a single neural layer with the first-order neighbors being considered. In order to allow for multi-hop neighbors, they proposed to stack multiple such neural layers. Consequently, the structural neighborhood information on graphs can be incorporated by a deep neural network architecture without explicitly parameterizing polynomials. Specifically, Equation (13) can be simplified to:

$$g_\theta *_G \mathbf{X} = \theta_0 \mathbf{X} - \theta_1 \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{X} \tag{14}$$

with two shared parameters $\theta_0$ and $\theta_1$. To reduce the number of parameters in practice, $\theta$ is used to replace $\theta_0$ and $\theta_1$ with $\theta = \theta_0 = -\theta_1$. Equation (14) can, in turn, be expressed as:

$$g_{\theta G}\mathbf{X} = \theta(\mathbf{I}_N + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})\mathbf{X} = \theta(\widehat{D}^{-\frac{1}{2}}\widehat{A}\widehat{D}^{-\frac{1}{2}})\mathbf{X} \tag{15}$$

where $\hat{A} = \mathbf{A} + \mathbf{I}_N$ and $\hat{D} = \sum_j \hat{A}_{ij}$ are the renormalized matrices of $\mathbf{A}$ and $\mathbf{D}$ to deal with exploding/vanishing gradient problems.

The second graph convolutional filter - diffusion convolutional network (DCN) filter also belongs to spectral convolutional operators, but this one is derived by modeling the traffic flow as a diffusion process, which is characterized by a Markov process (Li et al., 2017). The assumption is that after several time-steps, the diffusion process would stop and converge to a stationary distribution $\mathcal{P} \in \mathbb{R}^{N \times N}$. A $K$-step truncated stationary distribution $\mathcal{P}$ is used to characterize the transition probabilities between nodes. Additionally, a bidirectional diffusion process is included in this model such that the model can flexibly capture the impact of the traffic on both upstream and downstream roads. The resulting diffusion convolutional operation over graphs can be formulated as:

$$g_{\theta\ C}\mathbf{X} = \sum_{k=0}^{K-1} (\theta_{k,1}(\mathbf{D}_{\text{out}}^{-1}\mathbf{W})^k + \theta_{k,2}(\mathbf{D}_{\text{in}}^{-1}\mathbf{W}^T)^k)\mathbf{X} \tag{16}$$

where $\theta \in \mathbb{R}^{K \times 2}$ are the parameters of the bidirectional filter $g_\theta$, and $\mathbf{D}_{\text{out}}^{-1}\mathbf{W}$ and $\mathbf{D}_{\text{in}}^{-1}\mathbf{W}^{\mathbf{T}}$ are the state transition matrices of the diffusion process.

There are some similarities and differences between these two kinds of graph convolutions. Both GCN and DCN are designed from a spectral perspective and operate over non-Euclidean data structures, namely graphs. The difference is that GCN is defined on undirected graphs while DCN can be applied both on directed and undirected graphs. By introducing a similarity transformation, GCN can be considered as a special case of DCN.

In our proposed architecture, we can use either approach to capture spatial interactions over graphs, with the traffic condition $\mathbf{X}_{t-(M-1)}^{t}$ as the input. We want to investigate which model would work better with Transformer to model spatio-temporal dependencies.

## 4.3 | Traffic transformer architecture

Since Transformer itself can only deal with sequential information, it is unable to deal with complicated spatio-temporal dependencies. In order to address this problem, here we extend Transformer to *Traffic Transformer* by introducing a GNN. As illustrated in Figure 1, our proposed architecture conforms to an end-to-end sequence framework, composed of an encoder and a decoder.

In the encoder, a sequence of traffic $\mathcal{X}_{t-(M-1)}^{t}$ is first passed through a GCN layer by using Equation (15) or (16) to aggregate the neighborhood information on a road network, which captures the spatial dependency over nearby nodes. Then a fully connected neural network is employed to strengthen the expressiveness of the model. These features then are fed into the Transformer encoder cell to learn temporal features. A typical Transformer encoder cell is depicted in Figure 2. Note that this figure adopts elementwise addition-based combination. That is, each $H_{\text{GCN}}^{t}$ (in Figure 2) is added up with its temporal embedding so as to incorporate temporal information. As suggested in Section 4.1.3, we can also use similarity-based combination to account for temporal information. Then the results are fed through multi-head self-attention to aggregate the impact of the traffic at other time-steps on that at time-step $t$. Again, fully connected neural networks are employed to improve the expressiveness of the model.

As for the decoder, it has a similar structure except that the Transformer decoder cell has one more encoder–decoder attention block and another dense layer is leveraged before outputting the forecasts. More implementation details about Transformer encoder and decoder can be found in Vaswani et al. (2017). During the training process, a sequence of traffic $\mathcal{X}_{t}^{t+H-1}$ is fed into the decoder through the same series of neural networks as the input for the encoder and then is passed through the additional dense layer to map the outputs of the Transformer decoder cell to the predicted traffic $\mathcal{X}_{t+1}^{t+H}$. Unlike other recurrent models where the traffic data at different time-steps are fed recursively, the input sequence for the encoder/decoder can be sent to our architecture

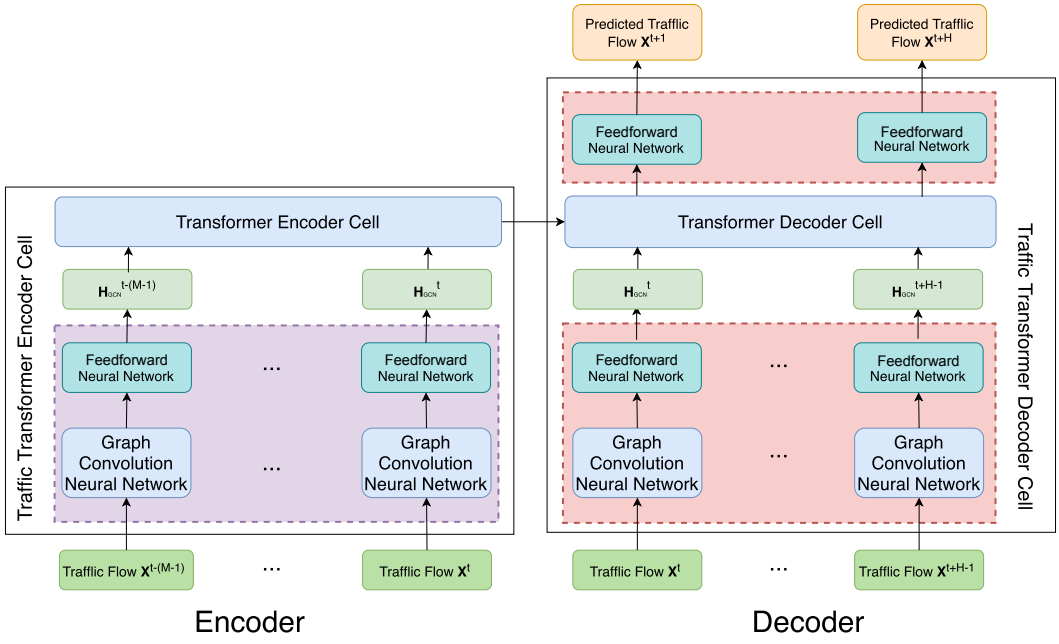



**FIGURE 1** Architecture of traffic transformer

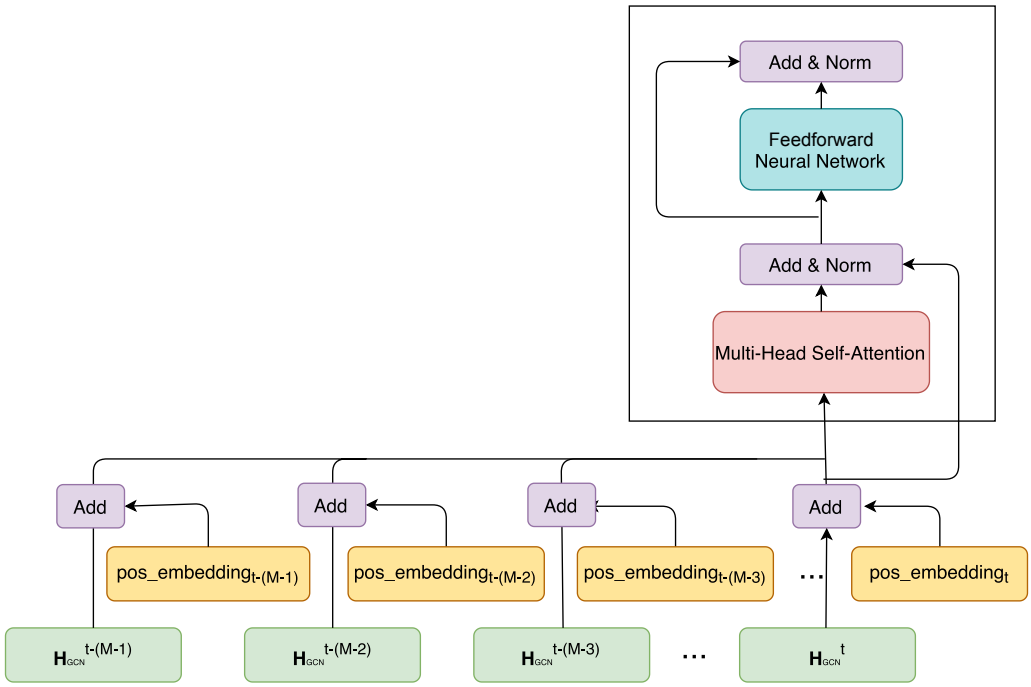



**FIGURE 2** Transformer encoder cell

simultaneously. Thus, the architecture is more computationally efficient during the training process. However, in the prediction phase, this decoder functions a little differently as any other model for time series prediction does, because in practice the input sequence for the decoder is unknown. In order to yield the predicted traffic at future time-steps, we use $\mathbf{X}^t$ as the first input for the decoder to output the predicted traffic $\mathbf{X}^{t+1}$, which then serves as

the next input for the decoder to gain $\mathbf{X}^{t+2}$. This whole process continues until the predicted horizon is met. Note that here $\mathbf{X}^{t}$ rather than a zero matrix, which is widely adopted by other models, is used as the first input for the decoder, since it is supposed to provide more useful information than zeros.

During the training phase, the aim of our model is to minimize the difference between the real and predicted traffic at each future time-step, measured by the mean absolute error (MAE). The loss function of Traffic Transformer can be formulated as:

$$Loss = \frac{1}{H}\sum_{t=1}^{H}\frac{1}{N}\sum_{j=1}^{N}|\mathbf{X}_j^t - \hat{X}_j^t| \tag{17}$$

where $\mathbf{X}_j^t$ is the predicted traffic expected to be observed at sensor $j$ at time $t$, and $\hat{X}_j^t$ is the corresponding ground truth.

## 5 | EXPERIMENTS

In this section we report on the experiments we carried out to evaluate the performance of the proposed architecture.

### 5.1 | Data set description

Two real-world benchmark data sets are used for evaluation: METR-LA (Li et al., 2017) from loop detectors in the highway of Los Angeles County and California Transportation Agencies Performance Measurement System (PeMS), respectively. Both data sets are sorted by time in ascending order (from the past to the present) and are split into three parts for training (70%), validation (10%), and testing (20%). Z-score normalization with the mean and standard derivation of the training data is applied to these three sets. To retain comparability with baselines, we use the sensor graphs of both data sets constructed by Li et al. (2017). Table 2 provides a basic description of each data set.

### 5.2 | Experimental details

#### 5.2.1 | Baselines

We compare our proposed Traffic Transformer with multiple baselines:

**TABLE 2** Basic description of two data sets

| Data set | METR-LA | PeMS-BAY |
|---|---|---|
| #sensors | 207 | 325 |
| region | Los Angeles County | Bay Area |
| time periods | March 1–June 30, 2012 | January 1–May 31, 2017 |
| #training_pairs | 23,974 | 36,465 |
| #validation_pairs | 3,425 | 5,209 |
| #testing_pairs | 6,850 | 10,419 |

1. Historical average (HA) model, which studies the seasonal trend of the traffic flow and then calculates the weighted average of seasonal traffic flow as the predictions (Liu & Guan, 2004).
2. Auto-regressive integrated moving average model with Kalman filter (ARIMA_kal), which is a typical parametric model in the time series community (Ahmed & Cook, 1979).
3. Linear support vector regression (LSVR) model (Smola & Schölkopf, 2004), which employs linear support vector machine to learn relationships between the input time series and the output time series from historical traffic flow and then predict the future traffic.
4. Feedforward neural network (FNN), composed of two dense layers with $L2$ normalization.
5. Fully connected LSTM (FC-LSTM, an RNN-based sequence model). This architecture conforms to an encoder–decoder framework composed of two LSTM layers on both the encoder and the decoder side (Sutskever et al., 2014).
6. Diffusion convolutional recurrent neural network (DCRNN), proposed by Li et al. (2017). As the name implies, it designs a diffusion convolutional operator by assuming that the traffic follows a diffusion process, and then this graph convolutional operator is fused with GRU to capture spatio-temporal dependencies.
7. Spatio-temporal graph convolutional networks (ST-GCN; see Yu et al., 2017), which is built entirely on spatial and temporal convolutional structures.

For more implementation details on these baselines, see Li et al. (2017). In order to maintain a fair comparison, we do not run most of the baselines but directly take the results from Li et al. (2017). For the ST-GCN model, we use open-source code provided by Yu et al. (2017) to train the models with the data sets we use in this article.

### 5.2.2 | Implementation details

To test the performance of different encoding methods, we set the length of a sequence in history, $M$, and the length of the forecasting sequence, $H$, to 12. For the time series segment method, the number of days for capturing weekly-periodic patterns, $w$, and daily-periodic patterns, $d$, are 1. The hidden dimension is 64 for all the layers in our model. Two encoder and decoder cells are used and one graph convolutional layer is utilized. We use open-source code to implement this entire architecture (https://github.com/tensorflow/models/tree/master/official/transformer). Additionally, as stated in Section 4.3, the forecasting sequence is unknown at prediction but known at training. As a result, there is a discrepancy between these two processes, which causes a quick error accumulation along the yielded sequence (Bengio, Vinyals, Jaitly, & Shazeer, 2015). In order to solve this problem, we also adopted scheduled sampling in our model to bridge the difference; see Bengio et al. (2015) for more details.

### 5.2.3 | Evaluation metrics

In order to evaluate and compare the performance of different models, we adopt the following three metrics: the MAE (see Equation 17); the root mean squared error (RMSE),

$$\mathrm{RMSE} = \sqrt{\frac{1}{H}\sum_{t=1}^{H}\frac{1}{N}\sum_{j=1}^{N}(\mathbf{X}_j^t - \hat{X}_j^t)^2} \tag{18}$$

and the mean absolute percentage error (MAPE),

$$\mathrm{MAPE} = \frac{1}{H}\sum_{t=1}^{H}\frac{1}{N}\sum_{j=1}^{N}\left|\frac{\mathbf{X}_j^t - \hat{X}_j^t}{\mathbf{X}_j^t}\right| \tag{19}$$

The first two metrics measure absolute prediction errors, while the last metric measures relative prediction errors. For all these metrics, smaller values mean better prediction performance. On both data sets, we exclude missing data when evaluating the performance of models.

## 5.3 | Experimental results

### 5.3.1 | Comparison of traffic prediction performance

Table 3 shows the prediction performance of different methods in terms of 15-, 30-, and 60-min-ahead prediction. Note that in this subsection we adopt the time series segment encoding method and similarity-based combination to implement our model, corresponding to TSE-SC in Table 4. Obviously, there are several interesting discoveries. In general, neural network-based models, including FNN, FC-LSTM, ST-GCN, DCRNN, and our model noticeably outperform these weak baselines which typically only focus on modeling temporal features. This indicates that these simple models are unable to capture the complicated spatio-temporal dependency in traffic. Furthermore, the prediction performance of Traffic Transformer is much better than the other models on both data sets in terms of all the evaluation metrics. Especially on the more challenging data set, METR-LA, the average prediction error is reduced by 26.8, 12.4, and 34.5% in terms of MAE, RMSE, and MAPE, respectively. These improvements demonstrate the effectiveness of our architecture in modeling complex spatio-temporal dependencies for traffic prediction.

**TABLE 3** Prediction performance of different models on two data sets

| T | Metric | HA | ARIMA_kal | SVR | FNN | FC-LSTM | ST-GCN | DCRNN | Our model (TSE-SC) |
|---|---|---|---|---|---|---|---|---|---|
| *METR-LA* | | | | | | | | | |
| | MAE | 4.16 | 3.99 | 3.99 | 3.99 | 3.44 | 3.8 | 2.77 | **2.43** |
| 15 min | RMSE | 7.8 | 8.21 | 8.45 | 7.94 | 6.3 | 7.9 | 5.38 | **4.73** |
| | MAPE | 13.00% | 9.60% | 9.30% | 9.90% | 9.60% | 9.48% | 7.30% | **6.57%** |
| | MAE | 4.16 | 5.15 | 5.05 | 4.23 | 3.77 | 5.07 | 3.15 | **2.79** |
| 30 min | RMSE | 7.8 | 10.45 | 10.87 | 8.17 | 7.23 | 10.28 | 6.45 | **5.61** |
| | MAPE | 13.00% | 12.70% | 12.10% | 12.90% | 10.90% | 12.90% | 8.80% | **7.45%** |
| | MAE | 4.16 | 6.9 | 6.72 | 4.49 | 4.37 | 7.16 | 3.6 | **3.28** |
| 60 min | RMSE | 7.8 | 13.23 | 13.76 | 8.69 | 8.69 | 13.43 | 7.59 | **6.68** |
| | MAPE | 13.00% | 17.40% | 16.70% | 14.00% | 13.20% | 13.45% | 10.50% | **9.08%** |
| *PEMS-BAY* | | | | | | | | | |
| | MAE | 2.88 | 1.62 | 1.85 | 2.2 | 2.05 | 1.46 | 1.38 | **1.22** |
| 15 min | RMSE | 5.59 | 3.3 | 3.59 | 4.42 | 4.19 | 3.24 | 2.95 | **2.78** |
| | MAPE | 6.80% | 3.50% | 3.80% | 5.19% | 4.80% | 3.01 | 2.90% | **2.76%** |
| | MAE | 2.88 | 2.33 | 2.48 | 2.3 | 2.2 | 1.94 | 1.74 | **1.59** |
| 30 min | RMSE | 5.59 | 4.76 | 5.18 | 4.63 | 4.55 | 4.27 | 3.97 | **3.61** |
| | MAPE | 6.80% | 5.40% | 5.50% | 5.43% | 5.20% | 4.59% | 3.90% | **3.43%** |
| | MAE | 2.88 | 3.38 | 3.28 | 2.46 | 2.37 | 2.52 | 2.07 | **1.77** |
| 60 min | RMSE | 5.59 | 6.5 | 7.08 | 4.98 | 4.96 | 5.52 | 4.74 | **4.36** |
| | MAPE | 6.80% | 8.3 | 8.00% | 5.89% | 5.70% | 6.05% | 4.90% | **4.29%** |

Bold values denote the best results.

**TABLE 4** Comparison of different temporal encoding methods in terms of RMSE

| | OPE | RPE | | GPE | | RPPE | | GPPE | | TSE | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | AC | AC | SC | AC | SC | AC | SC | AC | SC | AC | SC |
| 15 min | 5.75 | 5.26 | 5.07 | 7.47 | 5.10 | 8.47 | – | 8.42 | – | 4.91 | **4.73** |
| 30 min | 6.82 | 6.30 | 6.18 | 9.54 | 6.15 | 11.53 | – | 11.35 | – | 5.88 | **5.61** |
| 60 min | 8.79 | 7.45 | 7.30 | 12.57 | 7.32 | 13.99 | – | 14.08 | – | 6.91 | **6.68** |

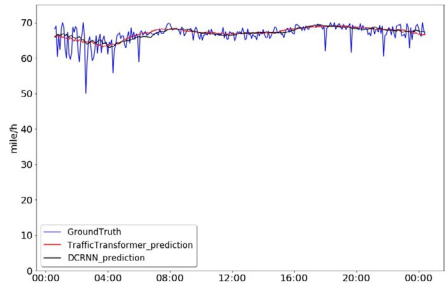
Bold values denote the best results.

In addition, to have a better understanding of why our model performs the best, we visualize the forecasting results of our model and the DCRNN model. From Figures 3a and b, we can observe that the DCRNN and our model achieve almost the same performance in the sense that both models are good at dealing with relatively stable traffic conditions. Figures 3c and d illustrates that our model has a better ability to accurately capture the abrupt changes in the traffic and the predicted traffic is aligned with the ground truth better than that of DCRNN. However, when the traffic changes very frequently and abruptly, as shown in Figures 3e and f, our model also struggles, although it is still better than DCRNN. More attention should be paid to this extreme circumstance in the future. In addition, we observe that our model and the DCRNN model yield completely different forecasting results between 22:00 and 2:00 in Figure 3g. Apparently, our prediction results are much more accurate. By comparing the similarity of the two forecasting time series at that time interval in this figure, we attribute our success to the ability of our model to capture periodic patterns.

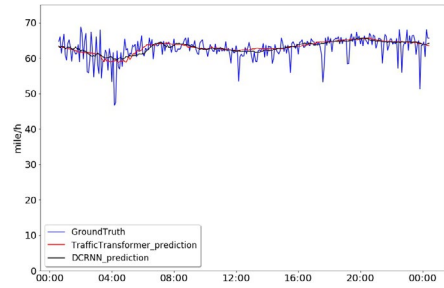
### 5.3.2 | Experimental comparison of different temporal encoding methods

With the aim of evaluating the effectiveness of our model in capturing temporal features, we compare the prediction performance of our models under different encoding methods. For most encoding methods, we consider two ways of injecting position embeddings into our models, namely addition-based combination (AC) and similarity-based combination (SC); see Section 4.1.3 for more details. However, we only apply AC to the encoding methods which are centered around the periodic position encoding strategy, since the similarity between the hybrid periodic position embedding and the position embedding of the future time-step is meaningless. In addition, we also include the results of using the original position encoding (OPE) strategy in Transformer. This experiment is conducted on the METR-LA data set, and only the RMSE is reported in Table 4.

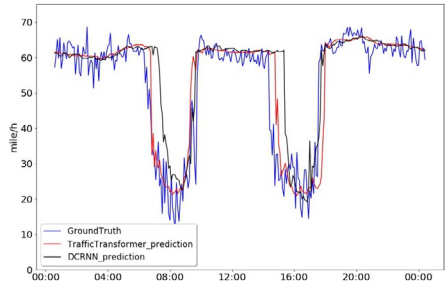
Table 4 describes the comparison results. We make the following observations. First, the encoding methods centered around the periodic position encoding strategy, including RPPE and GPPE, yield the worst results. This is because this idea in fact increases the difficulty of the model learning the commonality from training sequences, as more detailed temporal features in the position embedding reduce the similarity between two sequences. Second, the relative position encoding strategy (RPE) works much better than the global position encoding strategy (GPE) when addition-based combination is applied. This is led by the unseen position embeddings of the time-steps at testing. In the global position encoding strategy, we generate position indexes for all time-steps, and thus each position embedding is unique. The position embeddings of the time-steps in the testing set are never trained during the training process. However, when a similarity-based combination is adopted, they achieve similar results. This is attributed to the characteristics of Equation (5) as the similarity between different time-steps is only relevant to their time interval. Third, as expected, OPE has higher prediction errors. Since there is no translation relationship between the source and the target sequence in traffic forecasting, the original position encoding strategy is inapplicable to traffic forecasting. Finally, the best performance is shown by the TSE method and similarity-based combination shows a better result. The reasons why the TSE-SC method wins are two-fold.



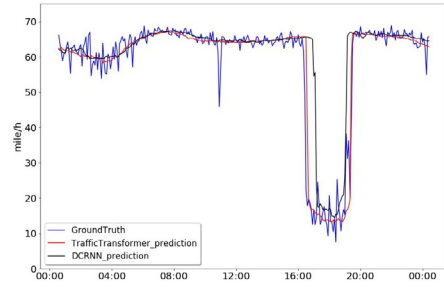

**(a)** Case 1: similar results to DCRNN



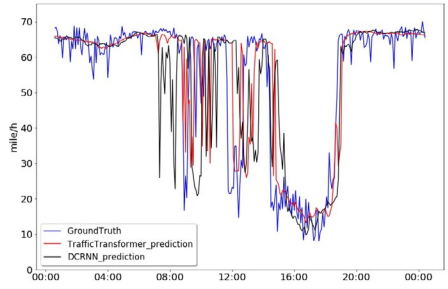

**(b)** Case 2: similar results to DCRNN



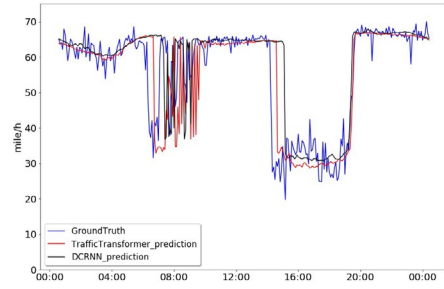

**(c)** Case 3: our model is better at capturing abrupt changes



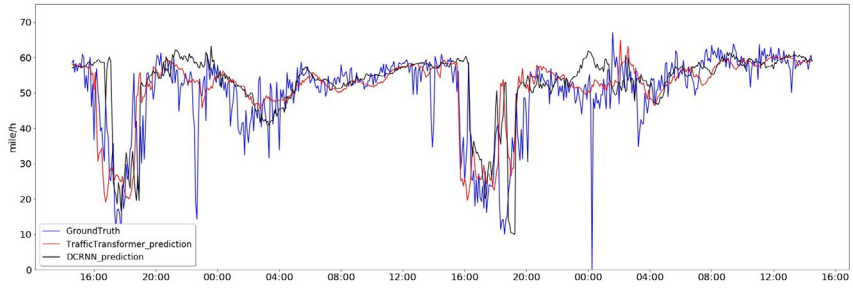

**(d)** Case 4: our model is better at capturing abrupt changes




**(e)** Case 5: both models need improving




**(f)** Case 6: both models need improving




**(g)** Case 7: our model benefits from the modeling of the periodicity of time series

**FIGURE 3** One-hour ahead traffic speed prediction on METR-LA data set

First, we explicitly enrich the input sequence with historical periodical information (daily and weekly segments), and thus these additional segments in fact enable the model to learn long temporal dependencies. Second, it is

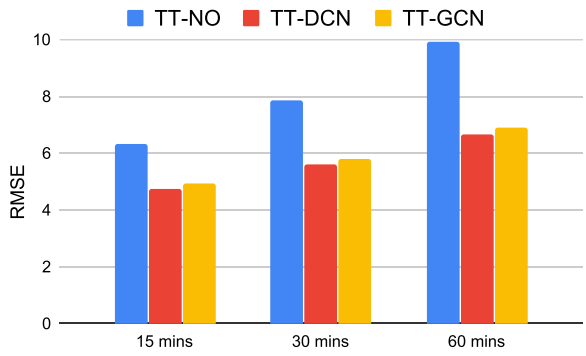



**FIGURE 4** Prediction results on METR-LA data set

attributed to the success of similarity-based combination when modeling temporal dependencies. By doing so, the contributions from different time-steps are modeled as similarity scores, as shown in Equations (10)–(12). The closer two time-steps are, the more important one time-step is to the other. Previous work in related fields has revealed that it is more appropriate to model time proximity as similarity scores to measure time-derived influence (Li, Cong, Li, Pham, & Krishnaswamy, 2015).

### 5.3.3 | Benefits of modeling spatial dependency

Aside from temporal dependencies, here we compare the performance of different graph convolutional operators in modeling spatial dependencies. Three variants of our model, Traffic Transformer (TT), are: TT-GCN, which utilizes the well-known GCN to capture spatial dependencies; TT-DCN, which that assumes the traffic flow follows a diffusion process; and TT-NO, which neglects the role of spatial dependencies.

Figure 4 shows the prediction performance of these variants with the same parameters. Without explicitly modeling the spatial dependency, TT-NO yields the worst results in terms of all the horizons. This effectively demonstrates the necessity of *explicitly* modeling the spatial dependency. Furthermore, TT-DCN consistently outperforms TT-GCN, though by a small margin. We believe this is because DCN can account for the direction of the traffic flow while GCN connot.

## 6 | CONCLUSIONS

In this research we have introduced the novel non-recurrent architecture called *Traffic Transformer*. We have successfully used it for traffic forecasting to capture spatio-temporal dependencies. This architecture can be regarded as an extension of Transformer, a well-known sequential model in natural language processing. In order to explore how to capture temporal dependencies in Transformer, we have proposed seven different ways of modeling the continuity and periodicity of time series. The time series segment method achieved the best result. Additionally, we introduced a graph convolutional neural network into Transformer for modeling the spatial dependencies of the traffic. By doing so, the dynamic spatio-temporal characteristics of traffic can be captured. Extensive experiments on two benchmark data sets show that our model is superior to the baselines, demonstrating the effectiveness of our proposed temporal encoding method and our proposed overall architecture. One limitation of this article is that we only account for the temporal attention mechanism, while different upstream roads at different times may contribute differently. In future work we will explore how to design a spatio-temporal attention mechanism to address this at a finer scale.

## ACKNOWLEDGMENTS

This work was partially supported by the NSF award 1936677, "Spatially-Explicit Models, Methods, and Services for Open Knowledge Networks", as well as Esri, Inc.

## ORCID

*Ling Cai* http://orcid.org/0000-0001-7106-4907
*Gengchen Mai* http://orcid.org/0000-0002-7818-7309
*Bo Yan* http://orcid.org/0000-0002-4248-7203

## REFERENCES

Ahmed, M. S., & Cook, A. R. (1979). Analysis of freeway traffic time-series data by using Box-Jenkins techniques. *Transportation Research Record*, *722*, 1–9.

Bengio, S., Vinyals, O., Jaitly, N., & Shazeer, N. (2015). Scheduled sampling for sequence prediction with recurrent neural networks. In C. Cortes, D. D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. *28*, pp. 1171–1179). Cambridge, MA: MIT Press.

Cao, X., Zhong, Y., Zhou, Y., Wang, J., Zhu, C., & Zhang, W. (2017). Interactive temporal recurrent convolution network for traffic prediction in data centers. *IEEE Access*, *6*, 5276–5289.

Cui, Z., Henrickson, K., Ke, R., & Wang, Y. (2019). Traffic graph convolutional recurrent neural network: A deep learning framework for network-scale traffic learning and forecasting. *IEEE Transactions on Intelligent Transportation Systems*, *21*, in press. 1–12.

Cui, Z., Ke, R., & Wang, Y. (2018). *Deep bidirectional and unidirectional LSTM recurrent neural network for network-wide traffic speed prediction*. International Workshop on Urban Computing (UrbComp). Preprint, arXiv:1801.02143.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., & Salakhutdinov, R. (2019). Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics* (pp. 2978–2988).Preprint, arXiv:1901.02860.

Davis, N., Raina, G., & Jagannathan, K. (2019). *Grids versus graphs: Partitioning space for improved taxi demand-supply forecasts*. Preprint, arXiv:1902.06515.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). *BERT: Pre-training of deep bidirectional transformers for language understanding*. Preprint, arXiv:1810.04805.

Fu, R., Zhang, Z., & Li, L. (2016). Using LSTM and GRU neural network methods for traffic flow prediction. In *Proceedings of the 31st Youth Academic Annual Conference of Chinese Association of Automation*, Wuhan, China (pp. 324–328). Piscataway, NJ: IEEE.

Guo, S., Lin, Y., Feng, N., Song, C., & Wan, H. (2019). Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*, Honolulu, HI (pp. 922–929). Menlo Park, CA: AAAI.

Hammond, D. K., Vandergheynst, P., & Gribonval, R. (2011). Wavelets on graphs via spectral graph theory. *Applied & Computational Harmonic Analysis*, *30*(2), 129–150.

Huang, W., Song, G., Hong, H., & Xie, K. (2014). Deep architecture for traffic flow prediction: Deep belief networks with multitask learning. *IEEE Transactions on Intelligent Transportation Systems*, *15*(5), 2191–2201.

Jin, W., Lin, Y., Wu, Z., & Wan, H. (2018). Spatio-temporal recurrent convolutional networks for citywide short-term crowd flows prediction. In *Proceedings of the Second International Conference on Compute and Data Analysis*, DeKalb, IL (pp. 28–35). New York, NY: ACM.

Ke, J., Zheng, H., Yang, H., & Chen, X. M. (2017). Short-term forecasting of passenger demand under on-demand ride services: A spatio-temporal deep learning approach. *Transportation Research Part C: Emerging Technologies*, *85*, 591–608.

Khandelwal, U., He, H., Qi, P., & Jurafsky, D. (2018). Sharp nearby, fuzzy far away: How neural language models use context. In *Proceedings of the 56th Meeting of the Association for Computational Linguistics*, Melbourne, Australia (Vol. *1*, pp. 284–294). Stroudsburg, PA: ACL.

Kipf, T. N., & Welling, M. (2016). *Semi-supervised classification with graph convolutional networks*. Preprint, arXiv:1609.02907.

Li, S., Jin, X., Xuan, Y., Zhou, X., Chen, W., Wang, Y.-X., & Yan, X. (2019). Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. In *Proceedings of the 32nd Conference on Neural Information Processing Systems*, Vancouver, BC, Canada (pp. 5244–5254). San Diego, CA: NIPS.

Li, X., Cong, G., Li, X.-L., Pham, T. -A. N., & Krishnaswamy, S. (2015). Rank-geoFM: A ranking based geographical factorization method for point of interest recommendation. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, Santiago, Chile (pp. 433–442). New York, NY: ACM.

Li, Y., & Moura, J. M. (2019). *Forecaster: A graph transformer for forecasting spatial and time dependent data*. Preprint, arXiv:1909.04019.

Li, Y., Yu, R., Shahabi, C., & Liu, Y. (2017). *Diffusion convolutional recurrent neural network: Data driven traffic forecasting*. Preprint, arXiv:1707.01926.

Lim, B., Arik, S. O., Loeff, N., & Pfister, T. (2019). *Temporal fusion transformers for interpretable multi-horizon time series forecasting*. Preprint, arXiv:1912.09363.

Liu, J., & Guan, W. (2004). A summary of traffic flow forecasting methods. *Journal of Highway & Transportation Research & Development*, *3*, 82–85.

Liu, W., Zheng, Y., Chawla, S., Yuan, J., & Xing, X. (2011). Discovering spatio-temporal causal interactions in traffic data streams. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Diego, CA (pp. 1010–1018). New York, NY: ACM.

Lv, Y., Duan, Y., Kang, W., Li, Z., & Wang, F.-Y. (2014). Traffic flow prediction with big data: A deep learning approach. *IEEE Transactions on Intelligent Transportation Systems*, *16*(2), 865–873.

Ma, X., Dai, Z., He, Z., Ma, J., Wang, Y., & Wang, Y. (2017). Learning traffic as images: A deep convolutional neural network for large-scale transportation network speed prediction. *Sensors*, *17*(4), 818.

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). *Improving language understanding by generative pre-training*. Retrieved from https://s3-us-west-2.amazonaws.com/openaiassets/researchcovers/languageunsupervised/language_understanding_paper.pdf

Shi, X., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-K., & Woo, W.-C. (2015). Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *Proceedings of the 28th Conference on Neural Information Processing Systems*, Montreal, Quebec, Canada (pp. 802–810). San Diego, CA: NIPS.

Smola, A. J., & Schölkopf, B. (2004). A tutorial on support vector regression. *Statistics and Computing*, *14*(3), 199–222.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Proceedings of the 27th Conference on Neural Information Processing Systems*, Bangkok, Thailand (pp. 3104–3112). San Diego, CA: NIPS.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ⋯ Polosukhin, I. (2017). Attention is all you need. In *Proceedings of the 31st Conference on Neural Information Processing Systems*, Long Beach, CA (pp. 5998–6008). San Diego, CA: NIPS.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., ⋯ Dean, J. (2016). *Google's neural machine translation system: Bridging the gap between human and machine translation*. Preprint, arXiv:1609.08144.

Wu, Y., Tan, H., Qin, L., Ran, B., & Jiang, Z. (2018). A hybrid deep learning based traffic flow prediction method and its understanding. *Transportation Research Part C: Emerging Technologies*, *90*, 166–180.

Xu, M., Dai, W., Liu, C., Gao, X., Lin, W., Qi, G.-J., & Xiong, H. (2020). *Spatial-temporal transformer networks for traffic flow forecasting*. Preprint, arXiv:2001.02908.

Yao, H., Tang, X., Wei, H., Zheng, G., & Li, Z. (2019). Revisiting spatial-temporal similarity: A deep learning framework for traffic prediction. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*, Honolulu, HI (pp. 3668–3675). Menlo Park, CA: AAAI.

Yu, B., Yin, H., & Zhu, Z. (2017). *Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting*. Preprint, arXiv:1709.04875.

Yu, B., Yin, H., & Zhu, Z. (2018). Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, Stockholm, Sweden (pp. 3634–3640). Vienna, Austria: IJCAI.

Zhang, J., Zheng, Y., Qi, D., Li, R., & Yi, X. (2016). DNN-based prediction model for spatio-temporal data. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, San Francisco, CA. New York, NY: ACM.

Zhao, L., Song, Y., Zhang, C., Liu, Y., Wang, P., Lin, T., ⋯ Li, H. (2019). T-GCN: A temporal graph convolutional network for traffic prediction. *IEEE Transactions on Intelligent Transportation Systems*, *21*, in press.

**How to cite this article:** Cai L, Janowicz K, Mai G, Yan B, Zhu R. Traffic transformer: Capturing the continuity and periodicity of time series for traffic forecasting. *Transactions in GIS*. 2020;00:1–20. https://doi.org/10.1111/tgis.12644