IET Intelligent Transport Systems

WILEY

IET The Institution of Engineering and Technology

ORIGINAL RESEARCH OPEN ACCESS

A Self-Correction Transformer Network for Traffic Flow Prediction Under Dynamic Spatio-Temporal Distributions

Jingru Sun[1] | Ziyu Qiu[1] | Yichuang Sun[2] | Oluyomi Simpson[2]

[1]College of Computer Science and Electronic Engineering, Hunan University, Changsha, China | [2]School of Physics, Engineering, and Computer Science, University of Hertfordshire, Hatfield, UK

Correspondence: Oluyomi Simpson (o.simpson@herts.ac.uk)

Received: 1 August 2024 | **Revised:** 23 April 2025 | **Accepted:** 2 May 2025

Funding: This work is supported by the following funding sources: the National Natural Science Foundation of China (62171182); the Natural Science Foundation of Hunan Province (2025JJ50345); the Natural Science Foundation Project, Chongqing Science and Technology Commission (CSTB2022NSCQMSX0770); the Science and Technology Plan Project, Hunan Provincial Department of Transportation (202306); and the Hunan Emergency Management Science and Technology Project (yjtkjxm_202407).

ABSTRACT

Precise and timely traffic flow prediction plays a critical role in developing intelligent transportation systems and has attracted considerable attention in recent decades. The traffic flow has a non-stationary character in both time and space, when the drift phenomenon appears, the traffic flow undergoes significant and sudden changes, bringing the challenge to the prediction. This paper proposed a self-supervised learning-based adaptive spatiotemporal self-correction transformer traffic flow prediction network (SCTNet). SCTNet can feel the drift with self-supervised learning, compute distribution features of the test data, obtain the distribution difference signal, feed it into the model as network correction information, and then adjust the spatiotemporal dependence of traffic flow adaptively to enhance prediction accuracy. The self-supervised learning method can adjust the model quickly and smoothly, and be utilized in most existing traffic flow prediction models. The experiments demonstrate that compared to existing models, the proposed self-supervised learning SCTNet has achieved state-of-the-art performance and exhibited strong adaptability to the dynamically changing spatiotemporal distributions of traffic data.

1 | Introduction

Accurate and timely traffic flow prediction is crucial for the successful implementation of intelligent transportation systems [1, 2]. It can assist road users in making informed travel decisions, alleviate traffic congestion, and improve traffic efficiency [3]. However, traffic prediction is highly challenging due to the inherent non-stationarity, non-linearity, and dynamics of ordinary time series data, as well as the complex spatial correlation of traffic flow.

To research the non-stationary character of traffic flow, some researchers started with mathematical models to simulate the traffic flow, including microscopic and macroscopic approaches, where macroscopic traffic flow models are of paramount importance to traffic surveillance and control. Macroscopic traffic flow models include the first-order and higher-order models. With the increasing complexity of road network structures and the consideration of influencing factors that cause non-stationary characteristics, factors such as weather and accidents have been taken into account in traffic flow modeling.

Compared with first-order models, higher-order models have a stronger modeling capability, therefore, higher-order macroscopic models have become the mainstream of traffic flow modelling, one example of such a model is the second-order

time–space-discretized traffic flow model [4], which has been applied to freeway traffic flow modelling [5], traffic state estimation, and prediction [6]. Such as Wang et al. proposed METANET models [7], model complex traffic flow dynamics in large-scale freeway networks with sufficient accuracy. Kang et al. proposed a fractional time-varying grey traffic flow model based on viscoelastic fluid [8], capturing the characteristics of a traffic flow system changing with time, achieving better stability and modeling effect.

However, the competence of a traffic flow model to reflect traffic reality is enabled by two things: the model structure and model parameters, once the structure and the parameters are determined, the model is determined and does not have the ability to self-correct. In non-stationary states, such as accidents, researchers employ multiple models and switch between different models, which is not preferable for real-time traffic control, and proposed a high requirement for the researcher's experience.

Regression-based traffic flow prediction algorithms can be categorized as data-driven or model-driven, with the latter using mathematical models such as AR [9], MA [10], ARMA [11], ARIMA [12], and SARIMA [13, 14] to capture traffic flow dynamics and make predictions. A rare advantage of above methods is their innate adaptability. In particular, all autoregressive models recalibrate their parameters upon the arrival of each batch of data. These methods rely on challenging assumptions in real-world scenarios, like stationarity and linearity. While differencing can tackle stationarity, excessive differencing risks information loss and complex models. Additionally, these methods model individual time series and lack generality in estimating traffic conditions across the entire road network.

The data-driven approach, utilizing historical data for prediction, benefits from machine learning techniques, such as support vector regression (SVR) [15], k-nearest neighbours (KNN) [16], and other statistical machine learning methods [17, 18]. These techniques significantly improve the accuracy of traffic flow forecasting. However, machine learning reliance on human experience for feature extraction [19], cannot be sufficient for handling large and complex data.

Deep learning technology can handle large-scale data and automatically extract relevant features and has emerged as a dominant approach for traffic flow prediction [20, 21], especially promoting the development of temporal and spatial feature extraction methods. For instance, STGCN [22] utilizes graph convolution and temporal convolution to forecast traffic flow, while graph attention graph convolutional network (GAGCN) [23] combines graph attention network (GAT) and graph convolutional network (GCN) to extract intricate spatial features and achieve precise prediction outcomes with the assistance of CNN. HMIAN [24] fuse other data, such as weather and holidays to improve the prediction accuracy. The spatiotemporal traffic flow prediction has shown promising results in enhancing prediction accuracy. GSTRGCT [25] integrates spatial panel modeling with deep learning via tensor decomposition, using DASWNN for adaptive spatial dependencies and autocorrelated attention for temporal patterns, achieving up to 62% and 59% performance gains on real-world traffic datasets.

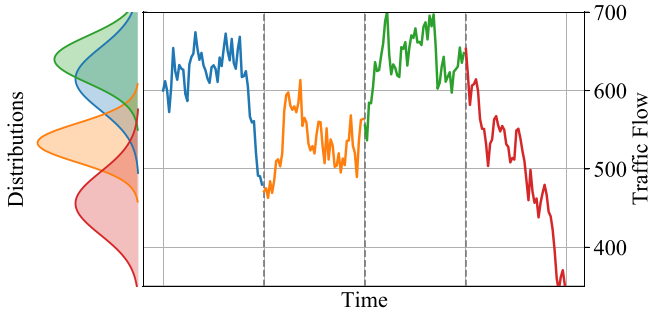


FIGURE 1 | The distribution of changes in traffic data makes predictions inaccurate.

The common structures of deep learning separate the training stage from the testing stage to capture spatiotemporal static features. However, traffic flow is an infinite and continuous unstable time series data collected by sensors, has the characteristics of data flow, as shown in Figure 1, daily average traffic flow data obtained from RTMC[1] (Regional Transportation Management Center, Minnesota Department of Transportation), showcasing variations in traffic flow distribution across different periods, predictions may become unreliable if the training data fails to encompass these patterns. While some methods employ unique network structures to extract non-stationary traffic or time series features and demonstrate a certain level of adaptability [26, 27], they still rely on training data and lack an effective mechanism to discern differences between training and testing data.

To tackle the challenge of predicting traffic flow in non-stationary situations, we propose an adaptive spatiotemporal traffic flow prediction model called the self-correction transformer network (SCTNet). This model is based on self-supervised learning and is designed to analyse and adjust predictions in real-time, close to the data source. Unlike traditional approaches that accumulate data to retrain the network, our method eliminates the need for retraining by continually processing the incoming data. The main contributions of this paper can be summarized as follows.

- SCTNet addresses the challenge of time-varying data distributions in traffic flow prediction by incorporating an adaptive enhancement module and a road network information fusion module within a seq2seq encoder–decoder framework. The model is modular and can be flexibly extended with alternative architectures or prior traffic knowledge to further boost prediction accuracy.
- SCTNet leverages a self-supervised learning paradigm by formulating auxiliary tasks that mitigate the limitations of traditional supervised learning. This design enables the model to adapt to distributional shifts in test data and iteratively refine its predictions through self-correction mechanisms.
- Extensive experiments on real-world traffic datasets demonstrate that SCTNet consistently outperforms state-of-the-art baselines in terms of accuracy and robustness. These results underscore the effectiveness and potential of SCTNet for practical deployment in intelligent transportation systems.

Compared to traditional spatiotemporal prediction models that often depend on periodic retraining and struggle to adapt to evolv-

ing traffic conditions, SCTNet offers a unified and adaptive framework capable of real-time adjustment to distributional changes in traffic data. Moreover, unlike prior approaches that typically decouple spatial and temporal feature modeling, SCTNet jointly encodes these dimensions via a transformer-based architecture, effectively capturing complex spatiotemporal interactions.

The remainder of this paper is organized as follows: Section 2 reviews the relevant literature and foundational techniques. Section 3 details the proposed SCTNet architecture and its underlying mechanisms. In Section 4, the model is evaluated on a real-world case-study road network, and the prediction results are validated. Finally, Section 5 summarizes the contributions and discusses potential future research directions.

2 | Related Work

2.1 | An Overview of Traffic Flow Predictions

Based on the discussion in Section 1, traffic flow prediction methods can be categorized into two groups: regression-based methods and simulation or analytical methods. Simulation-based methods, which are typically grounded in established traffic theories and principles [1, 13], excel at providing accurate predictions when the model aligns with objective facts. This characteristic distinguishes them from regression-based models that rely on training data. However, when faced with concept drift (distribution shift), the prediction accuracy simulation-based methods become ineffective.

Simulation-based methods include both macroscopic and microscopic approaches. Macroscopic methods analyse vehicle behaviour and interactions at a higher level of abstraction. In contrast, microscopic methods concentrate on predicting traffic flow based on individual roads, incorporating a more detailed analysis. Representative methods in this category encompass cellular automaton models [28] and car-following models [29]. Not only do these methods exhibit predictive capabilities, but they also provide insights for traffic condition control, owing to their robust theoretical foundation.

From a theoretical standpoint, they are not susceptible to concept drift. However, modelling the transportation system itself presents inherent challenges. The actual road conditions are diverse and constantly changing, leading to the complexity associated with these methods. For example, in well-known dynamic traffic assignment methods [30], the initial step involves constructing a dynamic origin–destination (OD) matrix. Utilizing microscopic methods, traffic flow changes are simulated based on road models, and the principles of game theory are introduced to simulate the strategy of vehicle road selection, aiming to achieve a system equilibrium. However, extending this approach to the entire road network incurs significant costs. Agent-based modelling (ABM) [31] simulates individual vehicle behaviours, capturing dynamic interactions that influence overall traffic flow. This method allows for a more granular understanding of traffic dynamics by considering the actions of each vehicle.

In the age of escalating data volumes, traditional regression-based methods have transitioned to deep learning techniques. These approaches are experiencing swift evolution, giving rise to various methodologies like [24, 32, 33]. Despite their progress, some challenges have been either overlooked or persistently unresolved. Among these challenges lies the accurate prediction of traffic flow in scenarios marked by shifting distributions.

2.2 | Self-Adaptive Prediction and Self-Supervised Learning

The Kalman filter [34] is the most well-known approach for self-adaptation. This recursive filter effectively reduces noise and estimates the system state based on incomplete information by efficiently sampling current observations. The concept of Kalman filtering has broad applications in navigation and control [35], signal processing [36], and even econometrics [37]. In traffic flow prediction [38], the Kalman filter is widely used. However, it assumes a linear system state space and Gaussian-distributed noise, not suitable for big data environments.

As mentioned in Section 1, statistical methods possess inherent adaptive capabilities. These methods, such as AR, MA, ARIMA, and SARIMA, determine the model order based on the input data and optimize the linear parameters of each component using statistical theories like maximum likelihood estimation. The above methods in the above approach start from basic mathematical assumptions and gradually build a comprehensive mathematical model. Among them, SARIMA is the most complex and has strong fitting capabilities for non-stationary periodic data. Reference [13] presents a traffic flow prediction model based on SARIMA specifically designed for weekly periodicity. Reference [14] provides a detailed comparison between SARIMA and various parametric and non-parametric prediction methods.

Currently, several adaptive algorithms based on deep learning have been proposed. Such as, in the field of fault detection, the approach in [39] applies adaptive ideas. Similarly, the method proposed in [40] implements a self-adaptive time series anomaly detection algorithm. Tonioni et al. [41] propose an online algorithm for continuous adaptive stereopsis that employs unsupervised learning to calculate errors and online back-propagation to the neural network. This approach aims to correct the sharp drop in the algorithm's accuracy caused by the significant difference between the usage scenario and the training data. An adaptive algorithm that also computes unsupervised loss and updates network parameters online is proposed in literature [42] for non-stationary time series forecasting. Framework Adarnn [26] tackles the distribution shift problem in time series forecasting by addressing temporal covariate shift (TCS) using adaptive algorithms for characterizing and matching temporal distributions, resulting in enhanced accuracy and minimized error.

Self-supervised learning has been widely used to train the network on unlabelled data and leverage auxiliary tasks to extract meaningful information. BERT [43] is a popular natural language processing model that learns the inherent nature of language by randomly masking words in the text and predicting them based on massive amounts of data. Several self-supervised learning tasks have been introduced in recent times. Literature [44] enables multi-sensor data representation through self-supervised learning. Another approach, described in [45], trains

an ensemble model comprising multiple LSTM neural networks, using unsupervised methods, to detect outliers in time series. Similarly, the authors in [46] have developed a time series change point detection algorithm based on the contrastive method in self-supervised learning.

2.3 | Advancements of SCTNet Based on Related Work

The non-stationary characteristics of time series are prevalent in traffic flow data and cannot be overlooked. While deep learning typically requires labelled data, labels for test data are often unavailable. Self-supervised learning addresses this issue by enabling the neural network to adapt without the need for explicit labels. Inspired by previous studies such as [41] and [42], SCTNet tackles the challenges posed by dynamic spatio-temporal distributions, improving algorithm performance. Building upon these insights, SCTNet introduces a sub-network structure that uses input data to calibrate prediction results in real-time. Unlike the mentioned methods, which often require frequent online updates, SCTNet avoids continuous network updates, simplifying implementation. This design choice ensures greater stability and eliminates concerns related to model degradation and catastrophic forgetting. As a result, SCTNet not only reduces computational complexity but also becomes more robust for deployment in real-world traffic scenarios.

3 | Methodology

3.1 | Problem Definition

We provide a formal definition of the spatiotemporal traffic flow prediction problem using a topology graph $G = (V, E, \mathbf{A})$. The adjacency matrix $\mathbf{A}$ captures the relationship between vertices, where the connection metric A_{ij} between vertices v_i and v_j can be a spatial distance or other similarity that may vary over time. The spatiotemporal graph $\mathbf{X}_{\mathcal{G}}^t$ represents the state of the road network, with c traffic flow feature data for N vertices generated at time t (e.g. primarily traffic speed, along with flow rate and occupancy) sampled at the same frequency for each detector.

For the spatiotemporal traffic flow prediction problem, the neural network takes a set of time-correlated graph signals $\mathcal{X}_{\mathcal{G}}^t = \left[\mathbf{X}_{\mathcal{G}}^{t-\tau-1}, \ldots, \mathbf{X}_{\mathcal{G}}^{t-1}\right]$ as input, and produces the prediction for the state at time t as $\mathcal{Y}_{\mathcal{G}}^t = \left[\hat{\mathbf{Y}}_{\mathcal{G}}^t, \ldots, \hat{\mathbf{Y}}_{\mathcal{G}}^{t+\tau'}\right]$, which can be expressed as either

$$\left[\mathbf{X}_{\mathcal{G}}^{t-\tau-1}, \ldots, \mathbf{X}_{\mathcal{G}}^{t-1}\right] \xrightarrow{f} \left[\hat{\mathbf{Y}}_{\mathcal{G}}^t, \ldots, \hat{\mathbf{Y}}_{\mathcal{G}}^{t+\tau'}\right] \tag{1}$$

or

$$\left[\hat{\mathbf{Y}}_{\mathcal{G}}^t, \ldots, \hat{\mathbf{Y}}_{\mathcal{G}}^{t+\tau'}\right] = f\left(\mathbf{X}_{\mathcal{G}}^{t-\tau-1}, \ldots, \mathbf{X}_{\mathcal{G}}^{t-1}, G\right), \tag{2}$$

where τ' represents the prediction length, τ represents the length of the time dimension of the input sequence (which is referred to as spatiotemporal graph in the previous context), and f refers to the function relationship that needs to be learned by the deep learning model.

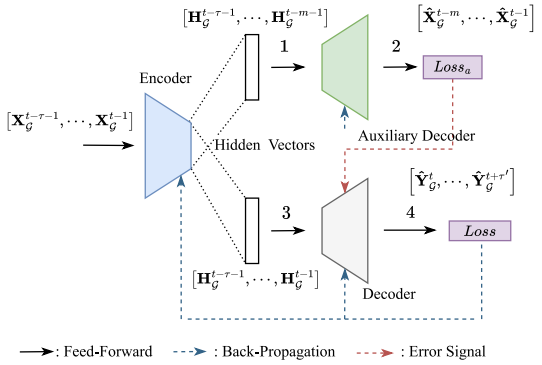


FIGURE 2 | The overall structure of SCTNet and the flow of data illustrated using the example of the prediction auxiliary task. The numbers indicate the execution steps of the algorithm.

3.2 | Overview of the Network

The SCTNet network framework is illustrated in Figure 2. It consists of an encoder, an auxiliary decoder, and a decoder, and the process involves training and testing phases.

During the training phase, the input data undergoes self-supervised learning based on the auxiliary task. When performing the prediction auxiliary task, the input data is separated into input features and labels. The encoder encodes input features and the auxiliary decoder decodes them to predict $\hat{Y}_a$. The loss is calculated as $\mathcal{L}_a = \hat{\mathbf{Y}}_a - \mathbf{Y}$. The term $\mathcal{L}_a$ refers to the difference between the traffic system features learned by the network and the current traffic data distribution. This difference is then used as an error signal to adjust the decoder. Next, the original input $\mathbf{X}$ is sent to the encoder-decoder network for prediction, resulting in $\hat{\mathbf{Y}}$. The model then calculates the prediction loss $\mathcal{L}$, which is used to update the parameters of the encoder, auxiliary decoder, and decoder. This process is described by the pseudocode in Algorithm 1.

During the inference stage, the algorithm performs self-supervised auxiliary tasks to obtain a loss value $\mathcal{L}_a$. This loss value allows the algorithm to sense dynamic changes in the transportation system. Next, the input data $\mathbf{X}$ is encoded, and the loss tensor $\mathcal{L}_a$ is sent to the decoder. The decoder then decodes the loss tensor to produce the corrected prediction value, $\hat{\mathbf{Y}}$, which is explained in detail in Algorithm 2.

The encoder and decoder structures, shown in Figure 5, are similar to the Transformer architecture. The encoder, located on the left side, learns the representation of input data. On the other hand, the decoder, placed on the right side, performs various prediction tasks and can also function as an auxiliary decoder. In the upcoming sections, we will introduce the design of the self-supervised task, along with the temporal and spatial feature extraction modules. Afterward, we will provide a thorough explanation of the encoder and decoder architecture,

ALGORITHM 1 | Training phase pseudocode.

Input: Training Dataset: D, Number of Iterations: I , Auxiliary Task Type: T , Mask Length: m
for $i \leftarrow 1$ **to** I **do**
  for $[\mathbf{X}_{\mathcal{G}}^{t-\tau-1}, \cdots, \mathbf{X}_{\mathcal{G}}^{t-1}], [\mathbf{Y}_{\mathcal{G}}^{t}, \cdots, \mathbf{Y}_{\mathcal{G}}^{t+\tau'}] \in \mathcal{D}$ **do**
    $\mathbf{X} \leftarrow [\mathbf{X}_{\mathcal{G}}^{t-\tau-1}, \cdots, \mathbf{X}_{\mathcal{G}}^{t-1}]$;
    $\mathbf{Y} \leftarrow [\mathbf{Y}_{\mathcal{G}}^{t}, \cdots, \mathbf{Y}_{\mathcal{G}}^{t+\tau'}]$;
    if $T =$ *Prediction* **then**
      $\mathbf{X}_a \leftarrow [\mathbf{X}_{\mathcal{G}}^{t-\tau-1}, \cdots, \mathbf{X}_{\mathcal{G}}^{t-m-1}]$;
      $\mathbf{Y}_a \leftarrow [\mathbf{X}_{\mathcal{G}}^{t-m}, \cdots, \mathbf{X}_{\mathcal{G}}^{t-1}]$;
    else if $T =$ *Reconstitution* **then**
      $\mathbf{X}_a \leftarrow [\mathbf{X}_{\mathcal{G}}^{t-\tau-1}, \cdots, \mathbf{X}_{\mathcal{G}}^{t-1}]$;
      $\mathbf{Y}_a \leftarrow [\mathbf{X}_{\mathcal{G}}^{t-1}, \cdots, \mathbf{X}_{\mathcal{G}}^{t-\tau-1}]$;
    end
    $\hat{\mathbf{Y}}_a = \mathbf{decoder}_a\left(\Omega_a; \mathbf{encoder}\left(\Theta; \mathbf{X}_a\right), [\mathbf{X}_a; \mathbf{X}_0]\right)$;
    $\mathcal{L}_a \leftarrow \mathbf{Loss}_a\left(\hat{\mathbf{Y}}_a, \mathbf{Y}_a\right)$;
    $\hat{\mathbf{Y}} = \mathbf{decoder}\left(\Omega; \mathbf{encoder}\left(\Theta; \mathbf{X}\right), [\mathbf{X}_a; \mathcal{L}_a]\right)$;
    $\mathcal{L} \leftarrow \mathbf{Loss}\left(\hat{\mathbf{Y}}, \mathbf{Y}\right)$;
    UpdateΩ,Θ and Ω_a;
  end
end
Output: Θ, Ω, Ω_a

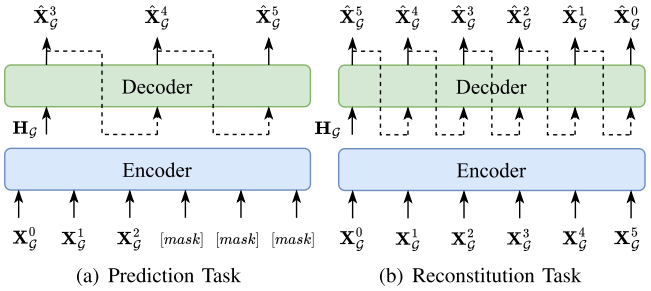
ALGORITHM 2 | Inference phase pseudocode.

Input: $[\mathbf{X}_{\mathcal{G}}^{t-\tau-1}, \cdots, \mathbf{X}_{\mathcal{G}}^{t-1}]$, Auxiliary Task Type: T, Mask Length: m
$\mathbf{X} \leftarrow [\mathbf{X}_{\mathcal{G}}^{t-\tau-1}, \cdots, \mathbf{X}_{\mathcal{G}}^{t-1}]$;
if $T =$ *Prediction* **then**
  $\mathbf{X}_a \leftarrow [\mathbf{X}_{\mathcal{G}}^{t-\tau-1}, \cdots, \mathbf{X}_{\mathcal{G}}^{t-m-1}]$;
  $\mathbf{Y}_a \leftarrow [\mathbf{X}_{\mathcal{G}}^{t-m}, \cdots, \mathbf{X}_{\mathcal{G}}^{t-1}]$;
else if $T =$ *Reconstitution* **then**
  $\mathbf{X}_a \leftarrow [\mathbf{X}_{\mathcal{G}}^{t-\tau-1}, \cdots, \mathbf{X}_{\mathcal{G}}^{t-1}]$;
  $\mathbf{Y}_a \leftarrow [\mathbf{X}_{\mathcal{G}}^{t-1}, \cdots, \mathbf{X}_{\mathcal{G}}^{t-\tau-1}]$;
end
$\hat{\mathbf{Y}}_a = \mathbf{decoder}_a(\Omega_a; \mathbf{encoder}(\Theta; \mathbf{X}_a), [\mathbf{X}_a; \mathbf{X}_0])$;
$\mathcal{L}_a \leftarrow \mathbf{Loss}_a\left(\hat{\mathbf{Y}}_a, \mathbf{Y}_a\right)$;
$\hat{\mathbf{Y}} = \mathbf{decoder}(\Omega; \mathbf{encoder}(\Theta; \mathbf{X}), [\mathbf{X}_a; \mathcal{L}_a])$;
Output: $\hat{\mathbf{Y}}$

which both include spatiotemporal feature extraction methods.

3.3 | Auxiliary Tasks Based on Self-Supervised Learning

Self-supervised learning aims to address the problem of limited labelled data by creating unsupervised auxiliary tasks. A straightforward approach presented in this paper involves dividing the input features into two segments. The first segment is used to predict the masked second segment, which results in a prediction error. This error is then used to guide the network to adjust the prediction. This approach offers a simple and direct way to improve the accuracy of the prediction. The second half sequence is masked and can be used as labelled data. This converts



(a) Prediction Task (b) Reconstitution Task

FIGURE 3 | Two strategies based on self-supervised learning: prediction task (a) and reconstruction task (b).

supervised learning into unsupervised learning (Figure 3a. This process can be formally represented as:

$$[\mathbf{X}_{\mathcal{G}}^{t-\tau-1}, \ldots, \mathbf{X}_{\mathcal{G}}^{t-\tau-1+m}] \xrightarrow{f'} [\hat{\mathbf{X}}_{\mathcal{G}}^{t-\tau+m}, \ldots, \hat{\mathbf{X}}_{\mathcal{G}}^{t-1}], \tag{3}$$

where m controls the input and label length of the auxiliary task. f' represents the learned function for the auxiliary task.

Another classic approach is the autoencoder [45, 47]. The basic idea is to encode the input graph signal sequence $\mathcal{X}_{\mathcal{G}}^{t}$ into a hidden space using a neural network and then decode it to obtain the reconstruction $\hat{\mathcal{X}}_{\mathcal{G}}^{t}$ of the input signal. The network is trained by comparing the deviation between the input sequence representing the "previous" state and the reconstruction signal representing the "current" state, to learn the distribution characteristics of the input signal. This is formally represented as follows,

$$[\mathbf{X}_{\mathcal{G}}^{t-\tau-1}, \ldots, \mathbf{X}_{\mathcal{G}}^{t-1}] \xrightarrow{f'} [\hat{\mathbf{X}}_{\mathcal{G}}^{t-1}, \ldots, \hat{\mathbf{X}}_{\mathcal{G}}^{t-\tau-1}]. \tag{4}$$

To prevent the network from simply outputting the input signal, the auxiliary task reconstructs the input data in reverse order. As shown in Figure 3b.

3.4 | Embedding Layer

Neural networks are suitable for processing low-dimensional, dense continuous features. To process traffic flow data, it is first fed into an embedding layer, which projects it into a low-dimensional space using a linear transformation.

$$\mathbf{E}' = \mathbf{W}[\mathbf{X}_{\mathcal{G}}^{t-\tau-1}, \ \ldots, \ \mathbf{X}_{\mathcal{G}}^{t-1}], \tag{5}$$

where $\mathbf{W} \in \mathbb{R}^{F \times F_{\text{in}}}$ and the same calculation is performed for each time step $t - i$, resulting in $\mathbf{E}' \in \mathbb{R}^{N \times \tau \times F_{\text{out}}}$.

The embedding layer uses time encoding to represent temporal relationships. Unlike RNNs that model sequential order directly, the SCTNet algorithm employs attention mechanisms to capture time series dependencies. However, attention mechanisms alone do not account for the temporal order of time steps. To resolve this issue, SCTNet incorporates sine-cosine time encoding, which is similar to the Transformer, to include temporal order information. The computation can be expressed as follows:

$$\mathbf{PE}(\text{pos}, 2i) = \sin\left(\text{pos}/\text{len}_m^{\,2i/F}\right), \tag{6}$$

$$\mathbf{PE}(\text{pos}, 2i+1) = \cos\left(\text{pos}/\text{len}_m^{\,2i/F}\right), \tag{7}$$

$$\mathbf{PE} = \mathbf{PE}(\text{pos}, 2i) \oplus \mathbf{PE}(\text{pos}, 2i+1), \tag{8}$$

where the position of each time step is represented by pos, and a parameter len_m is introduced to prevent cyclic repetition of time encoding. Typically, len_m is set to a large integer value, such as 10,000, based on the sequence length. The even and odd positions of the encoding vector are denoted by $2i$ and $2i+1$ respectively, and $\oplus$ represents concatenation. Equation (8) alternates between sine and cosine encoding to obtain the position encoding $\mathbf{PE} \in \mathbb{R}^{\tau \times F}$, where F is the feature dimension and τ is the sequence length. Finally, the position encoding is added to the feature obtained from the embedding layer:

$$\mathbf{E} = \mathbf{PE} + \mathbf{E}'. \tag{9}$$

3.5 | Temporal Feature Extraction Method

To enhance the accuracy of the model, SCTNet's spatial feature extraction module employs graph attention mechanisms. The model initially extracts time features and then combines them with features from other time steps. This integration of feature vectors for the complete input sequence during graph attention helps in improving the model accuracy.

In the temporal feature extraction layer, the algorithm uses a structure similar to transformer and informer [48]. The temporal feature extraction uses the famous scaled dot-product attention mechanism:

$$\mathbf{E}_{\text{t}} = \mathbf{W}_{\text{o}} \cdot \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\top}}{\sqrt{F}}\right)\mathbf{V}, \tag{10}$$

where $\mathbf{E}$ is the feature obtained through the embedding layer, the query vector $\mathbf{Q} = \mathbf{W}^{Q}\mathbf{E} \in \mathbb{R}^{N \times \tau \times F}$, the key vector $\mathbf{K} = \mathbf{W}^{K}\mathbf{E} \in \mathbb{R}^{N \times \tau \times F}$, the value vector $\mathbf{V} = \mathbf{W}^{V}\mathbf{E} \in \mathbb{R}^{N \times \tau \times F}$. Dividing the attention score by $\sqrt{F}$ helps to ensure numerical stability and convergence in the calculation. $\text{softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\Sigma_j \exp(z_j)}$ normalizes the attention score to obtain the attention distribution of the query vector. Introducing a multi-head attention mechanism can extract more complex features, and the complete calculation is as follows:

$$\begin{aligned}\mathbf{E}_{\text{t}} &= \text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \\ &= \mathbf{W}_{\text{o}} \cdot \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h),\end{aligned} \tag{11}$$

where $\text{head}_i = \text{softmax}(\frac{\mathbf{Q}_i\mathbf{K}_i^{\top}}{\sqrt{F/h}})\mathbf{V}_i \in \mathbb{R}^{\tau \times \frac{F}{h}} (i \in 1, \ldots, h)$ represents the features extracted by each attention head between time steps, $\mathbf{Q}_i = \mathbf{W}_i^{Q}\mathbf{E} \in \mathbb{R}^{\tau \times F/h}$, $\mathbf{K}_i = \mathbf{W}_i^{K}\mathbf{E} \in \mathbb{R}^{\tau \times F/h}$, $\mathbf{V}_i = \mathbf{W}_i^{V}\mathbf{E} \in \mathbb{R}^{\tau \times F/h}$, and $\text{Concat}(\cdot)$ represents concatenation operation. The calculation of each attention head head_i is consistent with Equation (10). They are concatenated in order and then subjected to a linear transformation using parameter matrix $\mathbf{W}_o$.

Due to the autoregressive nature required for modeling time series, the current time step can only be correlated with previous time steps. In attention-based models, the entire sequence $\mathbf{E}$ is input to the network at once. However, this approach includes future time steps when calculating attention scores, which can negatively affect the prediction performance. To address this issue, SCTNet uses a mask matrix $\mathbf{M}$ [49] to simulate causal convolution. This technique helps to improve prediction accuracy by limiting the model's knowledge of future time steps, as the mask matrix only allows the network to attend to past and present information. The mask matrix $\mathbf{M}$ is defined as follows:

$$\begin{pmatrix} 1 & -\text{inf} & \cdots & -\text{inf} \\ 1 & 1 & \cdots & -\text{inf} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{pmatrix}, \tag{12}$$

The matrix $\mathbf{M}$ is a lower triangular matrix used for masking. Each element $\mathbf{M}_{ij}$ of the matrix represents whether the time steps between i and j should be masked. If i is less than or equal to j, then $\mathbf{M}_{ij}$ is given a value of 1. Otherwise, it is assigned a value of $-\text{inf}$. This mask matrix is then multiplied element-wise with the attention score $\text{softmax}(\frac{\mathbf{Q}\mathbf{K}^{\top}}{\sqrt{F}})$ to generate the attention weights for each time step.

$$\begin{aligned}\mathbf{Attn} &= \text{softmax}\left(\mathbf{M} \odot \frac{\mathbf{Q}\mathbf{K}^{\top}}{\sqrt{F}}\right) \\ &= \begin{pmatrix} \alpha_{11} & 0 & \cdots & 0 \\ \alpha_{21} & \alpha_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{\tau 1} & \alpha_{\tau 2} & \cdots & \alpha_{\tau\tau} \end{pmatrix}.\end{aligned} \tag{13}$$

3.6 | Spatial Feature Extraction Method

The SCTNet model utilizes graph attention networks (GATs) to extract spatial features. The method used to calculate the multi-head graph attention network deviates from the one described in the literature [50]. To simplify implementation, a key-value attention calculation method similar to the one used in the time feature extraction layer is adopted. For example, let's consider a vertex $v \in G$ with its feature at time t denoted as $\mathbf{h}_v^t \in \mathbb{R}^F$. The attention mechanism calculates the similarity weight between $\mathbf{h}_v^t$ and other vertices $u \in V$ in the road network using this procedure:

$$s_{vu}^i = \frac{\mathbf{W}_i^{Q}\mathbf{h}_v^t\left(\mathbf{W}_i^{K}\mathbf{h}_u^t\right)^{\top}}{\sqrt{F/h}}, \tag{14}$$

where $s_{vu}^i \in \mathbb{R}$, i represents the index of attention heads, and h represents the total number of attention heads. For the entire road network, Equation 14 can also be written as:

$$\mathbf{S}^i = \frac{\mathbf{W}_i^{Q}\mathbf{H}^t\left(\mathbf{W}_i^{K}\mathbf{H}^t\right)^{\top}}{\sqrt{F/h}} \in \mathbb{R}^{N \times N}. \tag{15}$$

Afterward, the normalized correlation coefficients are computed for each column of $\mathbf{S}$ using the softmax function. β_{vu}^i represents

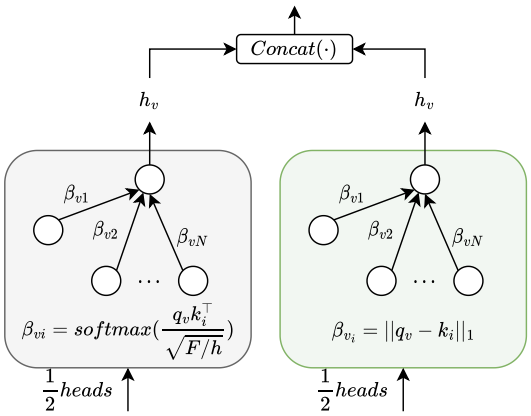


FIGURE 4 | Spatial feature extraction.

the similarity between vertices v and u for attention head i:

$$\beta_{vu}^{i} = \frac{\exp(s_{vu}^{i})}{\sum_{w \in \mathcal{N}i} \exp(s_{vw}^{i})}, \tag{16}$$

where $\mathcal{N}_v$ is the set of all neighbours of vertex v. The attention weights β computed by Equation (16) are used to generate the features of each vertex:

$$\mathbf{h}_v^i = \sum_{u \in \mathcal{N}_v} \beta_{vu} (\mathbf{W}_i^V \mathbf{h}_u^t), \tag{17}$$

where $\mathbf{W}_i^V \in \mathbb{R}^{N \times F/h}$ is the value vector transformation matrix. Since all vectors belong to the same time step in spatial attention calculation, the time step notation t is omitted for simplicity in the equation above.

The matrix calculation takes the following form:

$$\mathbf{h}_v' = \Big\Vert_{i=1}^{h} \sigma\left(\sum_{u \in \mathcal{N}_v} \beta_{vu}^i \mathbf{W}_i^V \mathbf{h}_u^i \right), \tag{18}$$

where the subscript or superscript i represents the number of attention heads, corresponding to different feature matrices $\mathbf{W}_i^V$. The design of multi-head attention mechanism can result in more complex feature representation of vertices. As can be seen from formulas (16) and (17), there are a large number of softmax calculations in the algorithm, which can slow down the computation speed.

Referring to the processing method in [51], SCTNet further optimizes the calculation of the spatial feature extraction layer. Here, SCTNet uses different similarity measurement algorithms to replace the softmax function in some attention heads to improve the efficiency of the algorithm. Therefore, when calculating the attention coefficients, SCTNet replaces half of the attention heads' similarity calculation with the 1-norm:

$$\beta_{vu}^{i} = ||\mathbf{q}_v^i - \mathbf{k}_u^i||_1, \tag{19}$$

where the $\mathbf{q}_v^i = \mathbf{W}_i^Q \mathbf{h}_v^i$ and $\mathbf{k}_u^i = \mathbf{W}_i^K \mathbf{h}_u^i$ in which $\mathbf{q}_v^i$ is the query matrix and $\mathbf{k}_u^i$ is the key matrix, as shown in Figure 4.

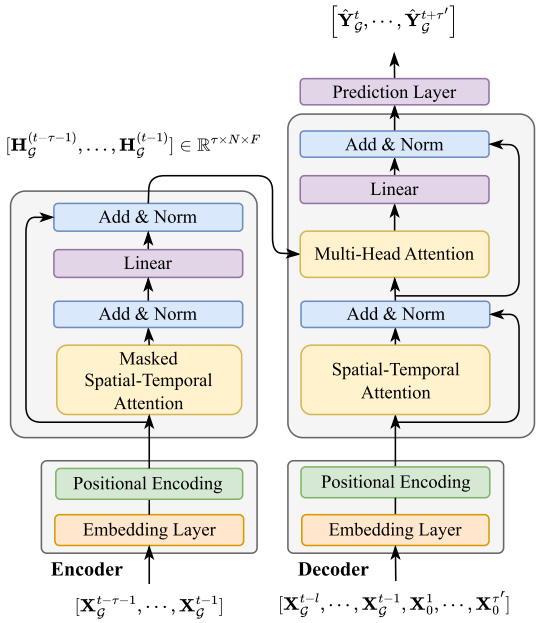


FIGURE 5 | Structure of encoder and decoder.

3.7 | SCTNet Encoder and Decoder

SCTNet uses a structure and method that closely resembles transformer, taking into account three factors. First, sequence processing is required for time series tasks, much like for translation tasks. Second, the transformer model has strong modeling capabilities. Third, unlike other encoder-decoder architectures, the transformer model's decoder has two inputs, making it a great tool to incorporate error information accurately into predictions.

The encoder consists of temporal and spatial feature extraction layers and a fully connected layer, and its detailed structure is shown in Figure 5. The input data is first projected into low-dimensional vectors by an embedding layer and combined with time encoding to obtain features **E**. Then, **E** is fed into the temporal feature extraction layer to extract temporal features by attention mechanism, and the results are further processed by the spatial feature extraction layer to extract spatial features. The temporal feature extraction layer and the spatial feature extraction layer can be encapsulated into a spatio-temporal feature extraction module (masked spatial–temporal attention in Figure 5). Similar to the attention mechanism, the results processed by the spatio-temporal feature extraction module are then sent to a fully connected layer for further processing, which also uses residual connections. This process also employs residual connections $F = x + \text{sublayer}(x)$ and layer normalization [49]. The encoder repeats the above operations multiple times to obtain the hidden state **H** of the input sequence after encoding:

The decoder, like the encoder, also uses a standard structure similar to the transformer. The decoder consists of a spatial-temporal attention module, a cross-attention module (multi-head attention in the decoder in Figure 5), and a fully connected layer, as shown in the decoder of Figure 5. The design of the decoder refers to the structure of the Informer network proposed in [48]. The decoder has two inputs, which are the output of the encoder **H** and a guiding vector. The guiding vector is composed of a

start token and an error signal tensor generated by an auxiliary task, and is sliced from the first $\tau - m$ time steps of the input sequence $\mathbf{X}$, denoted as $[\mathbf{X}_{\mathcal{G}}^{t-\tau-1}, \ldots, \mathbf{X}_{\mathcal{G}}^{t-m-1}]$. The guiding vector is generated differently in the auxiliary task stage and the prediction task stage:

$$\mathbf{X}_{de} = \left[\mathbf{X}_{\mathcal{G}}^{t-\tau-1}, \ldots, \mathbf{X}_{\mathcal{G}}^{t-m-1}\right] || \left[\mathbf{X}_0^1, \ldots, \mathbf{X}_0^m\right], \tag{20}$$

$$\mathbf{X}_{de} = \left[\mathbf{X}_{\mathcal{G}}^{t-\tau-1}, \ldots, \mathbf{X}_{\mathcal{G}}^{t-m-1}\right] || \left[\mathcal{L}_a^1, \ldots, \mathcal{L}_a^m\right], \tag{21}$$

$\mathbf{X}_{de} \in \mathbb{R}^{N \times (l+\tau')}$ is the guiding vector of the encoder. Equations (20) and (21) represent the guiding vectors used in the auxiliary task and final prediction task, respectively. $\left[\mathbf{X}_{\mathcal{G}}^{t-\tau-1}, \ldots, \mathbf{X}_{\mathcal{G}}^{t-m-1}\right]$ is the start token, $\mathbf{X}_0$ is a tensor sequence of the same length as the auxiliary task prediction, with all values set to 0, and $\mathcal{L}_a = \hat{\mathbf{Y}}_a - \mathbf{Y}_a$ represents the error information. $||$ represents the operation of concatenating vectors. The guiding vector goes through an embedding layer, time encoding, a spatial–temporal attention module, and a cross-attention layer for the attention mechanism calculation.

The decoder's cross-attention layer, also known as "multi-head attention," shown in Figure 5, is computed similarly to the time feature extraction layer. To put it simply, the query vector is derived from the guiding vector, while the key and value vectors are based on the input of the encoder. This enables better integration of the features obtained from the encoder and the self-supervised auxiliary task's error information into the prediction process. As $\mathbf{X}_{de}$ is not suitable for direct input to the network, another embedding layer with independent parameters is used to transform it and add time encoding to obtain $\mathbf{E}_{de}$. The resulting $\mathbf{E}_{de}$ is then used to generate query, key, and value vectors:

$$\mathbf{Q} = \mathbf{W}_i^Q \mathbf{E}_{de}, \mathbf{K} = \mathbf{W}_i^K \mathbf{E}_{en}, \mathbf{V} = \mathbf{W}_i^V \mathbf{E}_{en}, \tag{22}$$

where $\mathbf{E}_{en} = \mathbf{H}$ represents the encoded inputs.

Subsequently, the intermediate feature $\mathbf{E}$ is obtained by multiplying the similarity matrix $\mathbf{S}^i \in \mathbb{R}^{(\tau'+l)\times\tau}$ with $\mathbf{V}$ using Equation (10). The obtained $\mathbf{E}$ is then passed through a fully connected layer with residual connection and layer normalization, followed by a rectified linear unit (ReLU) activation. This process is repeated multiple times to extract spatio-temporal features, which are finally processed by the prediction layer to obtain the predicted results.

$$\hat{\mathbf{Y}}_{\mathcal{G}} = \mathbf{W}_2 \cdot \mathrm{ReLU}(\mathbf{W}_1 \mathbf{E} + \mathbf{b}_1) + \mathbf{b}_2, \tag{23}$$

where $\mathbf{W}_1 \in \mathbb{R}^{F \times F/2}$ and $\mathbf{W}_2 \in \mathbb{R}^{F/2 \times c}$ are learnable parameter matrices. The predicted values $\hat{\mathbf{Y}}$ of interest are obtained by slicing the tensor $\hat{\mathbf{Y}}_{\mathcal{G}}$: $\hat{\mathbf{Y}} = \hat{\mathbf{Y}}_{\mathcal{G}_{l+1:l+\tau'}}$.

The loss function measures the difference between predicted values and true values, which, in combination with the backpropagation algorithm, updates the network parameters. In this study, we use the Huber loss [52] (also known as SmoothL1 loss) as the loss function, which is defined as follows:

$$\mathcal{L}(\mathbf{Y}, \hat{\mathbf{Y}}) = \begin{cases} (\mathbf{Y} - \hat{\mathbf{Y}})^2/2 & \left|\mathbf{Y} - \hat{\mathbf{Y}}\right| \le \delta \\ \delta\left|\mathbf{Y} - \hat{\mathbf{Y}}\right| - \delta^2/2 & \text{otherwise} \end{cases}, \tag{24}$$

where δ is a threshold parameter that controls the range of the squared error loss. We choose the Huber loss because it is relatively insensitive to outliers, making it more effective in removing the influence of noise. The Huber loss uses the squared error when the prediction error is less than or equal to δ, while it uses the linear error when the prediction error is greater than δ.

4 | Experiments

4.1 | Performance Evaluation Metrics

Traffic flow prediction is a classic regression problem. The common metrics used to evaluate the performance of a regression model are mean absolute error (MAE), root mean square error (RMSE), and mMean absolute percentage error (MAPE).

4.2 | Datasets

PeMS-Bay: The PeMS-Bay dataset is a vehicle speed dataset derived from the PeMS system, which monitors and analyses real-time traffic conditions in California highways. It specifically covers traffic flow data in the San Francisco Bay Area, consisting of 325 sensors.

METR-LA: The dataset is collected from 207 loop detectors on highways in Los Angeles. The dataset contains vehicle speed data sampled from March 2012 to June 2012.

The experiment will conduct data cleaning on the two datasets mentioned above to perform linear interpolation on missing data. This is done to prevent errors in calculating MSE loss, which could cause weight updates of neural networks to fail. It also avoids division-by-zero errors when evaluating the MAPE error of the model. The statistics of the experimental datasets is given in Table 1.

To avoid exploding and vanishing gradients and enable the network to converge more quickly, the algorithm will perform RevIN instance normalization [53] on the data. The normalized data is compressed into the range of [0,1], which is beneficial for improving the stability and effectiveness of network training. The mean and variance of the data need to be preserved for later restoration, i.e. reverse normalization.

4.3 | Hyperparameter Settings

SCTNet consists of 8 key hyperparameters, including learning rate, batch size, epoch, encoder and decoder layer numbers, embedding dimension, attention head number, and dropout rate. To achieve optimal performance, a grid search is employed in this experiment to determine the best experimental parameters. The input sequence length is set to 12 time steps ($\tau = 12$), and the model predicates future traffic data for 3, 6, and 12 time steps ($\tau' = 3, 6, 12$), comparing prediction errors every 15, 30, and 60 min. During the training phase, the network parameters are optimized using the Adam optimizer in PyTorch. This optimizer is adaptive and adjusts the learning rate based on the loss. The initial learning rate of Adam is set to 1×10^{-4}, with betas set to

TABLE 1 | Statistics of experimental datasets.

Datasets	Sensors count	Sampling period	Timesteps	Means	Standard deviation
PeMS-Bay	325	2017/1/1 00:00:00–2017/6/30 23:55:00	52,115	62.61	9.59
METR-LA	207	2012/3/1 00:00:00–2012/6/27 23:55:00	34,272	53.71	20.26

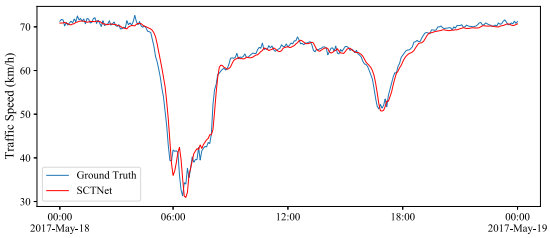


FIGURE 6 | The predicted fitting curve for detector 407157.

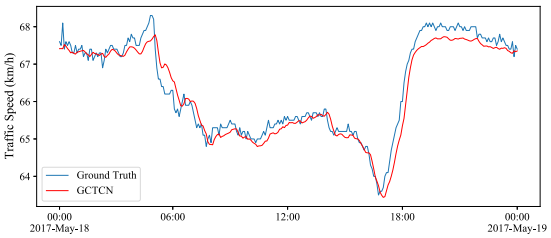


FIGURE 8 | The predicted fitting curve for detector 414282.

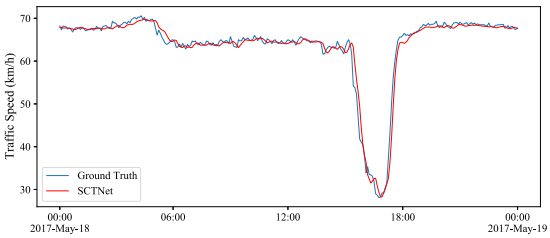


FIGURE 7 | The predicted fitting curve for detector 407187.

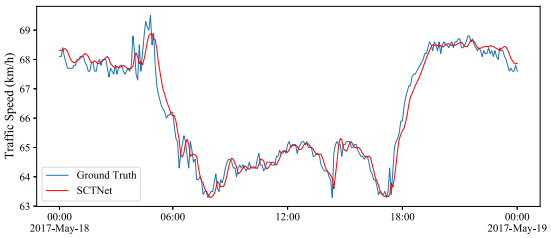


FIGURE 9 | The predicted fitting curve for detector 414694.

(0.9,0.999). The model is trained for 50 epochs, with a batch size of 16, an embedding dimension of 32, 4 encoder and decoder layers, 4 attention heads, and a dropout rate of 0.2. The loss function delta is set to 0.1. The historical average method utilizes the average value of input values from each time series as the prediction. The ARIMA model is best suited for non-stationary time series and serves as an important reference model. In comparison, SARIMA incorporates seasonal information and is more appropriate as a benchmark. However, it runs very slow for data with large seasonal periods. For this reason, we conducted separate comparative experiments using the PeMS-Downsampled dataset (derived from PeMS-Bay through downsampling), which still consists of 325 vertices.

In addition, tuning the parameters for a univariate ARIMA model can be challenging. In this experiment, we directly used the auto_arima function from the pmdarima[2] package to determine the parameters. This function combines the AIC (Akaike Information Criterion) and BIC (Bayesian Information Criterion) criteria to select the optimal model, ensuring more reliable experimental results while reducing the workload of parameter selection.

4.4 | Forecasting Performance

4.4.1 | Visualization of Predictions

The prediction results on the PeMS-Bay dataset for four randomly selected detector nodes are displayed in Figures 6 to 9. The SCTNet model's performance is relatively weaker during traffic peak periods, as it faces the challenge of predicting unpredictable traffic surges. However, the model's adaptive nature allows it to adjust and improve predictions when traffic data fluctuate. Despite variations in fitting accuracy among detectors, the SCTNet model effectively captures spatiotemporal features and trends in traffic flow changes at different locations. It successfully identifies the onset and end of peak periods, aligning with actual distribution patterns, aiding in predicting congestion, and explaining traffic phenomena.

4.4.2 | Comparison Experiment Results

Tables 2 and 3 present the comparison results between SCTNet and baseline models for different prediction horizons on PeMS-Bay and METR-LA.

Comparative analysis shows significant improvements in all metrics with SCTNet (utilizing the prediction auxiliary task), a mean increase of 10.1% in MAE accuracy, approximately 15.4% increase in MAPE, and an average increase of 27.0% in RMSE. Online error signals during testing adjust the dynamic weight relationship and learn spatial correlations.

Ablation experiments confirm the importance of the auxiliary task for prediction accuracy, but SCTNet still outperforms other models. However, it still outperforms other models, which confirms the effectiveness of the auxiliary task in enhancing the prediction accuracy. The self-supervised learning mechanism and attention-based spatiotemporal Transformer modules contribute to the improvement in predictive accuracy. Incorporating

TABLE 2 | Comparison of multi-step prediction errors on the PeMS-Bay dataset.

Model	MAE			RMSE			MAPE(%)		
	15 min	30 min	60 min	15 min	30 min	60 min	15 min	30 min	60 min
HA [54]	1.89	2.50	3.51	4.30	5.82	7.54	4.16	5.62	7.65
ARIMA [54]	1.62	2.33	3.38	3.30	4.76	6.50	3.50	5.40	8.30
VAR [55]	1.74	2.32	2.94	3.16	4.25	5.44	3.60	5.00	6.50
FNN [19]	2.20	2.30	2.46	4.42	4.63	4.98	5.19	5.43	5.89
SVR [56]	1.85	2.48	3.28	3.59	5.18	7.08	3.80	5.50	8.04
FC-LSTM [57]	2.05	2.20	2.37	4.19	4.55	4.96	4.80	5.20	5.70
DCRNN [58]	1.38	1.74	2.07	2.95	3.97	4.74	2.90	3.90	4.90
STGCN [22]	1.36	1.81	2.49	2.96	4.27	5.69	2.90	4.17	5.79
GMAN [32]	1.27	1.62	2.21	2.60	3.61	4.93	2.64	3.67	5.23
BTMF [59]	1.43	1.81	2.63	3.18	4.59	5.69	3.23	4.21	6.25
SCTNet[1]	1.37	1.74	1.98	2.53	3.64	4.72	2.67	3.91	4.57
SCTNet[2]	1.42	1.89	2.10	2.84	3.93	4.49	3.09	4.32	4.86

1The model using the prediction auxiliary task, and the following text is the same as this.
2The model using the reconstitution auxiliary task, and the following text is the same as this.

TABLE 3 | Comparison of multi-step prediction errors on the METR-LA dataset.

Model	MAE			RMSE			MAPE(%)		
	15 min	30 min	60 min	15 min	30 min	60 min	15 min	30 min	60 min
HA	4.79	5.47	6.99	10.00	11.45	13.89	11.70	13.50	17.54
ARIMA	3.99	5.15	6.90	8.21	10.45	13.23	9.60	12.70	17.40
VAR	4.42	5.41	6.52	7.80	9.13	10.11	10.2	12.70	15.80
FNN	3.99	4.23	4.49	7.94	8.17	8.69	9.90	12.90	14.00
SVR	3.39	5.05	6.72	8.45	10.87	13.76	9.30	12.10	16.70
FC-LSTM	3.44	3.77	4.37	6.30	7.23	8.69	9.60	10.09	14.00
DCRNN	2.77	3.15	3.60	5.38	6.45	7.60	7.30	8.80	10.50
STGCN	2.88	3.47	4.59	5.74	7.24	9.40	7.62	9.57	12.70
GMAN	2.85	3.30	3.91	5.29	6.35	7.54	7.46	9.37	11.68
BTMF	3.21	3.65	4.89	6.34	7.31	8.29	9.45	11.56	14.47
SCTNet[1]	2.68	3.19	3.52	5.65	6.60	7.46	8.29	9.95	11.06
SCTNet[2]	2.90	3.34	3.89	5.72	6.70	7.65	8.42	10.16	12.31

self-supervised enhancement techniques in training leads to faster and easier convergence and enhances the model's robustness.

Traditional machine learning algorithms, like SVR, are inferior to deep learning algorithms. Machine learning methods rely on data assumptions and struggle with large amounts of traffic flow data that spans a long period. In contrast, deep learning algorithms, such as FC-LSTM, can handle such data but have limitations in modeling spatial correlations compared to SVR. Graph neural network methods offer more versatility as they require training only one model instead of multiple LSTM or SVR models in parallel to cover all vertices.

STGCN, with its simple structure of causal and graph convolutional network with three layers, has limited capability in extracting long-term time features due to its restricted receptive field and shallow network depth. The predefined Laplacian matrix in GCN further limits the fixed receptive field of the convolution kernel. Consequently, STGCN falls behind SCTNet on datasets like PeMS-Bay and METR-LA.

Based on Sections 1 and 2.2, it is known that statistical-based methods require determining the orders and coefficients based on the training data before making predictions. Therefore, they have a natural adaptive property. A comprehensive experimental design must include them for comparison. The SARIMA model

TABLE 4 | Comparative experiment on PeMS-Downsampled dataset.

model	MAE	RMSE	MAPE (%)
ARIMA (auto, $T = 6$)	4.28	5.28	5.76
ARIMA (auto, $T = 12$)	4.81	6.67	7.74
SARIMA (auto, $T = 6$)	3.19	4.45	5.67
SARIMA (auto, $T = 12$)	3.37	5.27	6.69
SCTNet ($T = 6$)	2.40	4.29	4.58
SCTNet ($T = 12$)	2.94	5.29	5.67

The term "auto" in parentheses represents the usage of the auto_arima method to determine the parameters for each input. "$T = 6$" indicates a prediction length of 6. "SCTNet" refers to the model that incorporates auxiliary prediction tasks.

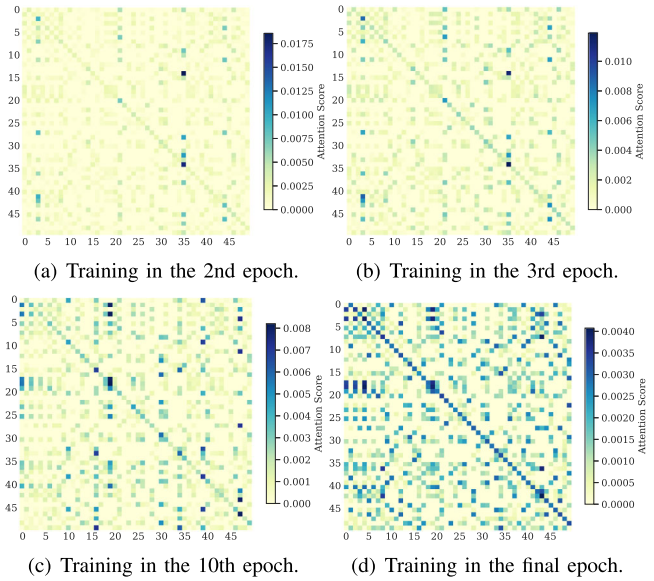
is not only suitable for non-stationary data but also capable of handling traffic flow data with clear periodicity. It is an ideal benchmark model. However, due to the high sampling frequency of the PeMS-Bay and METR-LA datasets, with a minimum period of 288, the SARIMA model struggles to handle such large periods. Therefore, in this study, we preprocess the PeMS dataset by downsampling it to a period of 6 and include corresponding comparative experiments to facilitate comparison with the SARIMA model.

Table 4 presents the comparison results between SCTNet and the benchmark ARIMA and SARIMA models on the PeMS-Downsampled dataset. Each input data (training data) has a length of 48 (Approximately 8 cycles, exceeding 1 week of data), which is sufficient for ARIMA and SARIMA. It can be observed that the statistical models perform worse than SCTNet, as discussed earlier. Additionally, SARIMA and ARIMA are univariate time series models, requiring the determination of orders for each sequence in the road network, making the process cumbersome. Their generality is noticeably inferior to deep learning-based spatio-temporal traffic flow models. Furthermore, the automatic determination of (p, d, q) and (P, D, Q) for each training data segment using the "auto_arima" function from the pmdarima package is a slow process and has slower computational speed compared to all deep learning models.

Figures 10 and 11 illustrate the evolution process of the spatial attention matrix and the temporal attention coefficients at different time steps of the model on the PeMS-Bay dataset, indirectly demonstrating the adaptive ability of the algorithm. Figure 10 shows the spatial attention matrix during the training phase (only the attention weights of the first 50 vertices are displayed), revealing that the model gradually extracts spatial correlation information from the data during the training process. As shown in Figure 11, the model can adaptively adjust the weights between time steps for different input sequences, dynamically extracting temporal relevance.

4.4.3 | Validation of Adaptive Capabilities

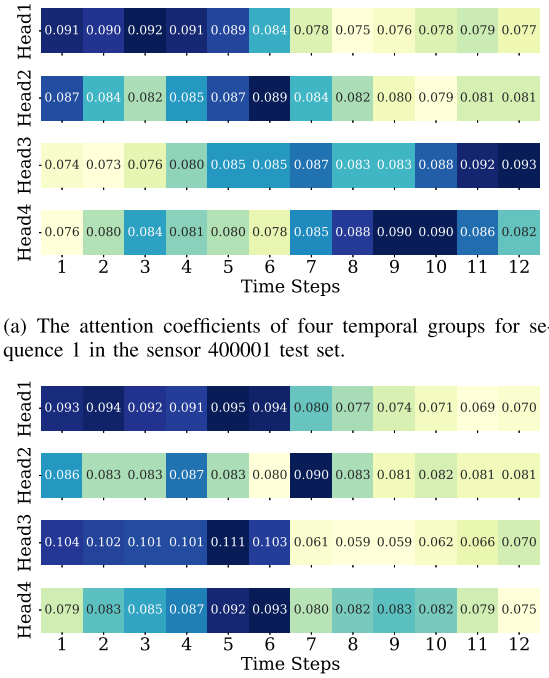
The experiment selected four sets of non-stationary data, 400211, 400227, 414282, and 414694 from the PeMS-Bay dataset, as shown



(a) Training in the 2nd epoch. (b) Training in the 3rd epoch.

(c) Training in the 10th epoch. (d) Training in the final epoch.

FIGURE 10 | The weight values of attention head 1 undergo a changing process.

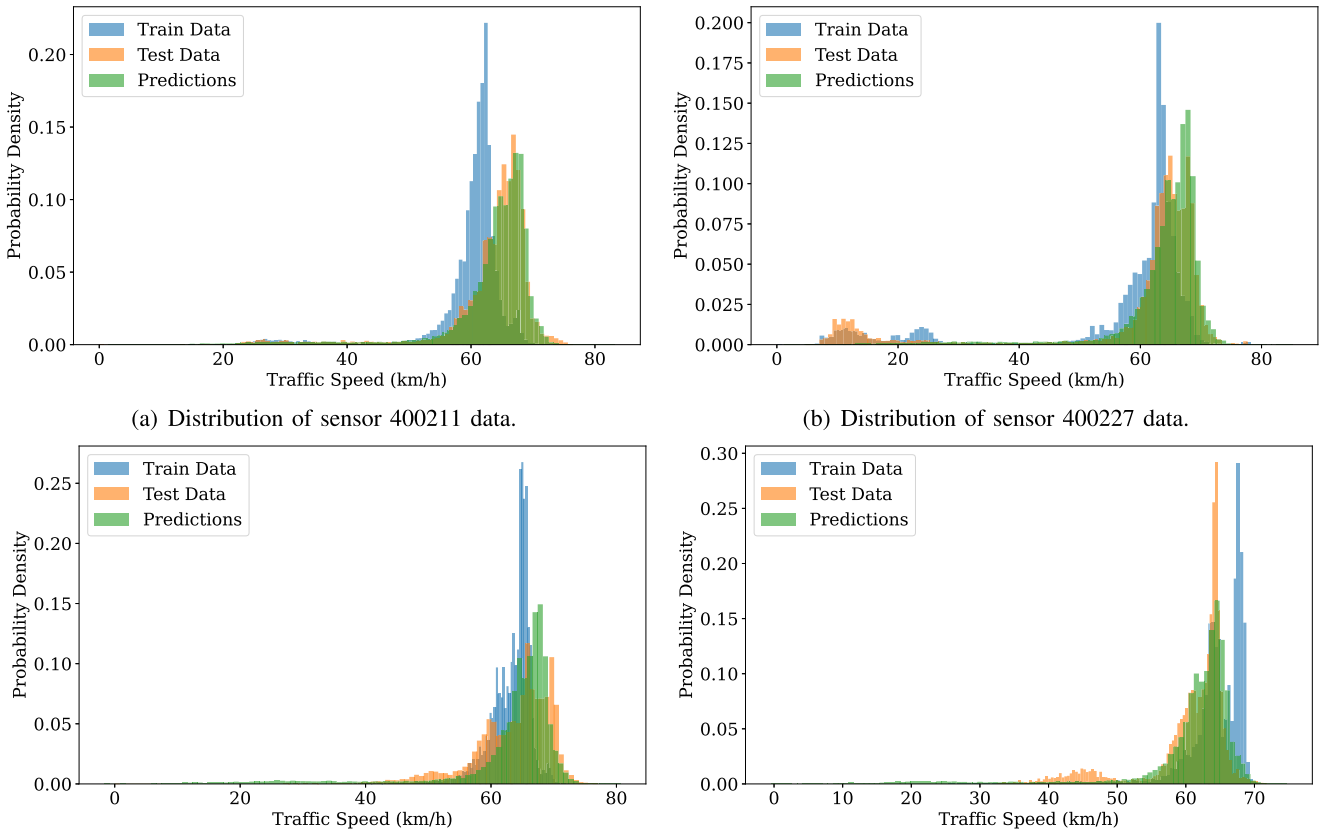


(a) The attention coefficients of four temporal groups for sequence 1 in the sensor 400001 test set.

(b) The attention coefficients of four temporal groups for sequence 100 in the sensor 400001 test set.

FIGURE 11 | The temporal attention coefficients in dataset 400001.

in Figure 12. It can be seen from the figure that the mean and variance of the training data of 400211 are relatively small, and the distribution of the testing data and the training data are significantly different, indicating the occurrence of concept drift. The prediction results of SCTNet show that the distribution of the predicted data is basically consistent with that of the testing data, which further proves the adaptive prediction ability of the SCTNet model. This feature is conducive to improving the prediction accuracy. The experimental results of other sensors also support this claim.



(a) Distribution of sensor 400211 data.

(b) Distribution of sensor 400227 data.

(c) Distribution of sensor 414282 data.

(d) Distribution of sensor 414694 data.

FIGURE 12 | Probability distributions of training set, testing set, and predicted values.

TABLE 5 | Ablation experiment results.

Dataset	Model	MAE	RMSE	MAPE (%)
PeMS-Bay (60 min)	Transformer-GAT	2.21	4.67	5.19
	SCTNet[1]	1.98	4.72	4.57
	SCTNet[2]	2.10	4.49	4.86
METR-LA (60 min)	Transformer-GAT	4.01	7.92	12.64
	SCTNet[1]	3.52	7.46	11.06
	SCTNet[2]	3.89	8.42	12.31

4.4.4 | Results of Ablation Study

To evaluate the significance of each module, ablation experiments were conducted. The baseline model, transformer-GAT, shares the same encoder-decoder structure as SCTNet but lacks self-supervised auxiliary tasks. As shown in Table 5, the prediction auxiliary task significantly improves performance, likely due to its alignment with traffic flow prediction, capturing both temporal and spatial dependencies. The reconstruction task also contributes but with a smaller effect, possibly due to its more general nature and the absence of a consistent ensemble model. Regardless of the self-supervised task, SCTNet consistently outperforms transformer-GAT, demonstrating that self-supervised learning enhances adaptability to non-stationary features in time-series data, boosting predictive power and robustness.

The prediction task aligns closely with traffic flow forecasting, both focusing on estimating future states. In contrast, the reconstruction task targets data feature consistency, not directly optimizing future prediction accuracy, which may limit its impact. Theoretically, the prediction task's loss function minimizes future deviations, aligning with the goal of traffic flow prediction. In comparison, the reconstruction task focuses on global data consistency without sufficiently addressing temporal dependencies.

We also use the T-test to test the significance of our model in 1 h ahead prediction compared to GAGCN. The p-value is less than 0.05, which demonstrates that SCTNet statistically outperforms GAGCN.

5 | Conclusion

The non-stationarity in the spatiotemporal distribution of traffic flow proposed a challenge to traffic flow prediction, how to adjust the model quickly and smoothly to improve prediction accuracy remains a hard problem. In this paper, we proposed an adaptive SCTNet adjustment method to handle the non-stationarity state. First, SCTNet employs self-supervised auxiliary tasks for estimating input state distribution and differences in traffic flow distribution states, generating an error signal. Second, the error signals together with input data feed into a spatio-temporal traffic flow prediction model to complete traffic flow prediction. in this paper, the prediction model is an encoder-decoder graph attention mechanism model, which extracts

spatial and temporal features from the error signal and input data to make predictions. The experimental results demonstrate that SCTNet exhibits adaptive capabilities, effectively capturing non-stationary characteristics and fine-tuning the model when facing slight changes in traffic data distribution, to improve the prediction accuracy and robustness. The self-supervised auxiliary task can apply to most of the existing spatiotemporal traffic flow prediction models and has a good university.

SCTNet has several limitations: First, the training complexity and computational cost are high due to the need to train two separate networks—one for prediction and another for the self-supervised task—which increases the computational burden, particularly with large-scale traffic networks. Second, while self-supervised tasks improve adaptability and robustness, designing appropriate tasks for different traffic data types remains a challenge. Lastly, although SCTNet adapts well to minor shifts in data distribution, it struggles with extreme or sudden traffic changes, which can lead to a decrease in prediction accuracy.

Future work is focused on designing better self-supervised auxiliary tasks, developing robust models for handling data distribution shifts, and addressing the issue of concept drift using online incremental algorithms. Additionally, exploring self-supervised auxiliary tasks based on road network topology for spatial correlation is a promising direction. Further testing and verification in diverse datasets and scenarios are necessary to enhance the reliability and applicability of the SCTNet model.

Author Contributions

Jingru Sun: conceptualization, formal analysis, funding acquisition, methodology, resources, validation, investigation, software. **Ziyu Qiu:** original draft preparation, validation, formal analysis, software. **Yichuang Sun:** project administration, supervision, writing original draft preparation. **Oluyomi Simpson:** visualization, supervision, writing – original draft preparation, investigation.

Acknowledgements

This work is supported by the following funding sources: the National Natural Science Foundation of China (62171182); the Natural Science Foundation of Hunan Province (2025JJ50345); the Natural Science Foundation Project, Chongqing Science and Technology Commission (CSTB2022NSCQMSX0770); the Science and Technology Plan Project, Hunan Provincial Department of Transportation (202306); and the Hunan Emergency Management Science and Technology Project (yjtkjxm_202407).

Conflicts of Interest

The authors have no conflicts of interest.

Data Availability Statement

Data sharing not applicable to this article as no datasets were generated or analysed during the current study.

Endnotes

[1] https://www.d.umn.edu/~tkwon/TMCdata/TMCarchive.html

[2] https://pypi.org/project/pmdarima/

References

1. B. S. Kerner, *Introduction to Modern Traffic Flow Theory and Control* (Springer, 2009).

2. X. Shen, J. Gao, W. Wu, M. Li, C. Zhou, and W. Zhuang, "Holistic Network Virtualization and Pervasive Network Intelligence for 6G," *IEEE Communications Surveys & Tutorials* 24, no. 1 (2021): 1–30.

3. J. Barros, M. Araujo, and R. J. Rossetti, "Short-Term Real-Time Traffic Prediction Methods: A Survey," in *2015 International Conference on Models and Technologies for Intelligent Transportation Systems (MTITS)* (IEEE, 2015), 132–139.

4. A. Messner and M. Papageorgiou, "METANET: A Macroscopic Simulation Program for Motorway Networks," *Traffic Engineering & Control* 31, no. 8–9 (1990): 466–470.

5. A. Kotsialos, M. Papageorgiou, C. Diakaki, Y. Pavlis, and F. Middelham, "Traffic Flow Modeling of Large-Scale Motorway Networks Using the Macroscopic Modeling Tool METANET," *IEEE Transactions on Intelligent Transportation Systems* 3, no. 4 (2002): 282–292.

6. Y. Wang, M. Zhao, X. Yu, et al., "Real-Time Joint Traffic State and Model Parameter Estimation on Freeways With Fixed Sensors and Connected Vehicles: State-of-the-Art Overview, Methods, and Case Studies," *Transportation Research Part C: Emerging Technologies* 134 (2022): 103444.

7. Y. Wang, X. Yu, J. Guo, et al., "Macroscopic Traffic Flow Modelling of Large-Scale Freeway Networks With Field Data Verification: State-of-the-Art Review, Benchmarking Framework, and Case Studies Using METANET," *Transportation Research Part C: Emerging Technologies* 145 (2022): 103904.

8. Y. Kang, S. Mao, and Y. Zhang, "Fractional Time-Varying Grey Traffic Flow Model Based on Viscoelastic Fluid and its Application," *Transportation Research Part B: Methodological* 157 (2022): 149–174.

9. X. Xu, X. Jin, D. Xiao, C. Ma, and S. Wong, "A Hybrid Autoregressive Fractionally Integrated Moving Average and Nonlinear Autoregressive Neural Network Model for Short-Term Traffic Flow Prediction," *Journal of Intelligent Transportation Systems* 27, no. 1 (2023): 1–18.

10. B. M. Williams, P. K. Durvasula, and D. E. Brown, "Urban Freeway Traffic Flow Prediction: Application of Seasonal Autoregressive Integrated Moving Average and Exponential Smoothing Models," *Transportation Research Record* 1644, no. 1 (1998): 132–141.

11. C. Zheng and L. Li, "The Improvement of the Forecasting Model of Short-Term Traffic Flow Based on Wavelet and ARMA," in *2010 8th International Conference on Supply Chain Management and Information* (IEEE, 2010), 1–4.

12. G. Yu and C. Zhang, "Switching ARIMA Model Based Forecasting for Traffic Flow," in *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing* (IEEE, 2004), ii–429.

13. B. M. Williams and L. A. Hoel, "Modeling and Forecasting Vehicular Traffic Flow as a Seasonal ARIMA Process: Theoretical Basis and Empirical Results," *Journal of Transportation Engineering* 129, no. 6 (2003): 664–672.

14. B. L. Smith, B. M. Williams, and R. Keith Oswald, "Comparison of Parametric and Nonparametric Models for Traffic Flow Forecasting," *Transportation Research Part C: Emerging Technologies* 10, no. 4 (2002): 303–321, https://doi.org/10.1016/S0968-090X(02)00009-8.

15. W. C. Hong, Y. Dong, F. Zheng, and C. Y. Lai, "Forecasting Urban Traffic Flow by SVR With Continuous ACO," *Applied Mathematical Modelling* 35, no. 3 (2011): 1282–1291.

16. L. Zhang, Q. Liu, W. Yang, N. Wei, and D. Dong, "An Improved K-Nearest Neighbor Model for Short-term Traffic Flow Prediction," *Procedia - Social and Behavioral Sciences* 96 (2013): 653–662, https://doi.org/10.1016/j.sbspro.2013.08.076.

17. X. Dong, T. Lei, S. Jin, and Z. Hou, "Short-Term Traffic Flow Prediction Based on XGBoost," in *2018 IEEE 7th Data Driven Control and Learning Systems Conference (DDCLS)* (IEEE, 2018), 854–859.

18. Y. Liu and H. Wu, "Prediction of Road Traffic Congestion Based on Random Forest," in *2017 10th International Symposium on Computational Intelligence and Design (ISCID)* (IEEE, 2017), 361–364.

19. Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," *Nature* 521, no. 7553 (2015): 436–444.

20. Y. Wu, H. Tan, L. Qin, B. Ran, and Z. Jiang, "A Hybrid Deep Learning Based Traffic Flow Prediction Method and its Understanding," *Transportation Research Part C: Emerging Technologies* 90 (2018): 166–180.

21. Y. Tian, K. Zhang, J. Li, X. Lin, and B. Yang, "LSTM-Based Traffic Flow Prediction With Missing Data," *Neurocomputing* 318 (2018): 297–305.

22. B. Yu, H. Yin, and Z. Zhu, "Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting," *arXiv:1709.04875* (2017).

23. C. Tang, J. Sun, Y. Sun, M. Peng, and N. Gan, "A General Traffic Flow Prediction Approach Based on Spatial-Temporal Graph Attention," *IEEE Access* 8 (2020): 153731–153741.

24. J. Sun, M. Peng, H. Jiang, Q. Hong, and Y. Sun, "HMIAN: a Hierarchical Mapping and Interactive Attention Data Fusion Network for Traffic Forecasting," *IEEE Internet of Things Journal* 9, no. 24 (2022): 25685–25697.

25. L. Xiong, L. Su, S. Zeng, X. Li, T. Wang, and F. Zhao, "Generalized Spatial–Temporal Regression Graph Convolutional Transformer for Traffic Forecasting," *Complex & Intelligent Systems* 10, no. 6 (2024): 7943–7964, https://doi.org/10.1007/s40747-024-01578-x.

26. Y. Du, J. Wang, W. Feng, et al., "AdaRNN: Adaptive Learning and Forecasting of Time Series," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management* (ACM, 2021), 402–411.

27. L. Bai, L. Yao, C. Li, X. Wang, and C. Wang, "Adaptive Graph Convolutional Recurrent Network for Traffic Forecasting," *Advances in Neural Information Processing Systems* 33 (2020): 17804–17815.

28. M. Fukui and Y. Ishibashi, "Traffic Flow in 1D Cellular Automaton Model Including Cars Moving With High Speed," *Journal of the Physical Society of Japan* 65, no. 6 (1996): 1868–1870.

29. P. G. Gipps, "A Behavioural Car-Following Model for Computer Simulation," *Transportation Research Part B: Methodological* 15, no. 2 (1981): 105–111.

30. M. Brackstone and M. McDonald, "Car-Following: A Historical Review," *Transportation Research Part F: Traffic Psychology and Behaviour* 2, no. 4 (1999): 181–196.

31. H. Chen and H. A. Rakha, "Multi-Step Prediction of Experienced Travel Times Using Agent-Based Modeling," *Transportation Research Part C: Emerging Technologies* 71 (2016): 108–121, https://doi.org/10.1016/j.trc.2016.07.004.

32. C. Zheng, X. Fan, C. Wang, and J. Qi, "GMAN: A Graph Multi-Attention Network for Traffic Prediction," *Proceedings of the AAAI Conference on Artificial Intelligence* 34, no. 1 (2020): 1234–1241.

33. L. Zhao, Y. Song, C. Zhang, et al., "T-GCN: A Temporal Graph Convolutional Network for Traffic Prediction," *IEEE Transactions on Intelligent Transportation Systems* 21, no. 9 (2019): 3848–3858.

34. S. J. Julier and J. K. Uhlmann, "New Extension of the Kalman Filter to Nonlinear Systems," in *Signal Processing, Sensor Fusion, and Target Recognition VI* (International Society for Optics and Photonics, 1997), 182–193.

35. T. Yucelen and A. J. Calise, "Kalman Filter Modification in Adaptive Control," *Journal of Guidance, Control, and Dynamics* 33, no. 2 (2010): 426–439.

36. M. Roth, G. Hendeby, C. Fritsche, and F. Gustafsson, "The Ensemble Kalman Filter: A Signal Processing Perspective," *EURASIP Journal on Advances in Signal Processing* 2017, no. 1 (2017): 1–16.

37. A. C. Harvey, *Forecasting, Structural Time Series Models and the Kalman Filter* (Cambridge University Press, 1990).

38. J. Guo, W. Huang, and B. M. Williams, "Adaptive Kalman Filter Approach for Stochastic Short-Term Traffic Flow Rate Prediction and Uncertainty Quantification," *Transportation Research Part C: Emerging Technologies* 43 (2014): 50–64.

39. W. Mao, J. Chen, X. Liang, and X. Zhang, "A New Online Detection Approach for Rolling Bearing Incipient Fault via Self-Adaptive Deep Feature Matching," *IEEE Transactions on Instrumentation and Measurement* 69, no. 2 (2020): 443–456, https://doi.org/10.1109/TIM.2019.2903699.

40. A. A. Cook, G. Mısırlı, and Z. Fan, "Anomaly Detection for IoT Time-Series Data: A Survey," *IEEE Internet of Things Journal* 7, no. 7 (2020): 6481–6494, https://doi.org/10.1109/JIOT.2019.2958185.

41. A. Tonioni, F. Tosi, M. Poggi, S. Mattoccia, and L. D. Stefano, "Real-Time Self-Adaptive Deep Stereo," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (IEEE, 2019), 195–204.

42. S. O. Arik, N. C. Yoder, and T. Pfister, "Self-Adaptive Forecasting for Improved Deep Learning on Non-Stationary Time-Series," *arXiv:2202.02403* (2022).

43. J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding," *arXiv:1810.04805* (2018).

44. A. Saeed, F. D. Salim, T. Ozcelebi, and J. Lukkien, "Federated Self-Supervised Learning of Multisensor Representations for Embedded Intelligence," *IEEE Internet of Things Journal* 8, no. 2 (2021): 1030–1040, https://doi.org/10.1109/JIOT.2020.3009358.

45. T. Kieu, B. Yang, C. Guo, and C. S. Jensen, "Outlier Detection for Time Series With Recurrent Autoencoder Ensembles," in *IJCAI'19: Proceedings of the 28th International Joint Conference on Artificial Intelligence* (ACM, 2019), 2725–2732.

46. S. Deldari, D. V. Smith, H. Xue, and F. D. Salim, "Time Series Change Point Detection With Self-Supervised Contrastive Predictive Coding," in *Proceedings of the Web Conference 2021* (ACM, 2021), 3124–3135.

47. P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, "LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection," *arXiv:1607.00148* (2016).

48. H. Zhou, S. Zhang, J. Peng, et al., "Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting," in *Proceedings of the AAAI Conference on Artificial Intelligence* 35 (2021), 11106–11115.

49. A. Vaswani, N. Shazeer, N. Parmar, et al., "Attention is all you Need," in *Advances in Neural Information Processing Systems*, (eds. I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett) (Curran Associates, 2017).

50. P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph Attention Networks," *arXiv:1710.10903* (2017).

51. C. Chen, C. Tao, and N. Wong, "LiteGT: Efficient and Lightweight Graph Transformers," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management* (ACM, 2021), 161–170.

52. P. J. Huber, "Robust Estimation of a Location Parameter," in *Breakthroughs in Statistics* (Springer, 1992), 492–518.

53. T. Kim, J. Kim, Y. Tae, C. Park, J. H. Choi, and J. Choo, "Reversible Instance Normalization for Accurate Time-Series Forecasting against Distribution Shift," in *International Conference on Learning Representations* (ICLR, 2022), 1–25.

54. M. Peixeiro, *Time Series Forecasting in Python* (Manning, 2021).

55. Z. Lu, C. Zhou, J. Wu, H. Jiang, and S. Cui, "Integrating Granger Causality and Vector Auto-Regression For Traffic Prediction of Large-Scale WLANs," *KSII Transactions on Internet and Information Systems (TIIS)* 10, no. 1 (2016): 136–151.

56. A. J. Smola and B. Schölkopf, "A Tutorial on Support Vector Regression," *Statistics and Computing* 14 (2004): 199–222.

57. R. Li, T. Zhong, X. Jiang, G. Trajcevski, J. Wu, and F. Zhou, "Mining Spatio-Temporal Relations via Self-Paced Graph Contrastive Learning," in *KDD '22: Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Association for Computing Machinery, 2022), 936–944.

58. Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting," *arXiv:1707.01926* (2017).

59. X. Chen and L. Sun, "Bayesian Temporal Factorization for Multidimensional Time Series Prediction," *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, no. 9 (2022): 4659–4673, https://doi.org/10.1109/TPAMI.2021.3066551.