When Do Contrastive Learning Signals Help Spatio-Temporal Graph Forecasting?

Xu Liu[1*], Yuxuan Liang[1*], Chao Huang[2], Yu Zheng[3,4], Bryan Hooi[1], Roger Zimmermann[1]
[1]School of Computing, National University of Singapore, Singapore
[2]Department of Computer Science, University of Hong Kong, China
[3]JD Intelligent Cities Research, China [4]JD iCity, JD Technology, China
{liuxu, yuxliang, bhooi, rogerz}@comp.nus.edu.sg; chaohuang75@gmail.com; msyuzheng@outlook.com

ABSTRACT

Deep learning models are modern tools for spatio-temporal graph (STG) forecasting. Though successful, we argue that data scarcity is a key factor limiting their recent improvements. Meanwhile, contrastive learning has been an effective method for providing self-supervision signals and addressing data scarcity in various domains. In view of this, one may ask: can we leverage the additional signals from contrastive learning to alleviate data scarcity, so as to benefit STG forecasting? To answer this question, we present the first systematic exploration on incorporating contrastive learning into STG forecasting. Specifically, we first elaborate two potential schemes for integrating contrastive learning. We then propose two feasible and efficient designs of contrastive tasks that are performed on the node or graph level. The empirical study on STG benchmarks demonstrates that integrating graph-level contrast with the joint learning scheme achieves the best performance. In addition, we introduce four augmentations for STG data, which perturb the data in terms of graph structure, time domain, and frequency domain. Experimental results reveal that the model is not sensitive to the proposed augmentations' semantics. Lastly, we extend the classic contrastive loss via a rule-based strategy that filters out the most semantically similar negatives, yielding performance gains. We also provide explanations and insights based on the above experimental findings. Code is available at https://github.com/liuxu77/STGCL.

CCS CONCEPTS

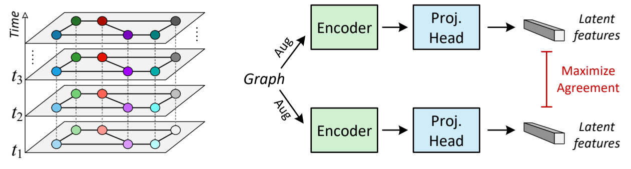
• **Information systems → Spatial-temporal systems**.

KEYWORDS

Spatio-temporal graph, contrastive learning, deep neural network

ACM Reference Format:
Xu Liu[1*], Yuxuan Liang[1*], Chao Huang[2], Yu Zheng[3,4], Bryan Hooi[1], Roger Zimmermann[1] . 2022. When Do Contrastive Learning Signals Help Spatio-Temporal Graph Forecasting?. In *The 30th International Conference on Advances in Geographic Information Systems (SIGSPATIAL '22), November 1–4, 2022, Seattle, WA, USA*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3557915.3560939

*The first two authors contributed equally to this work.



(a) Spatio-temporal graph (b) Contrastive learning on graphs

Figure 1: Illustration of STG and contrastive learning. The latent features can be either node [43] or graph [34] features. Aug: augmentation. Proj: projection.

1 INTRODUCTION

Deploying a large number of sensors to perceive an urban environment is the basis for building a smart city. The time-varying data that are produced from the distributed sensors can usually be represented as a spatio-temporal graph (STG), as shown in Figure 1(a). Leveraging the collected data, one important application is to forecast future trends based on historical observations, such as traffic forecasting. State-of-the-art approaches for this problem typically use convolutional neural networks (CNN) [7, 8, 17, 27, 32] or recurrent neural networks (RNN) [1, 4, 18, 23] to model temporal dependencies. For capturing spatial correlations, these methods mostly utilize the popular Graph Neural Networks (GNN) [15, 28].

While tremendous efforts have been made to design sophisticated architectures to capture complex spatio-temporal correlations, we argue that *data scarcity* is an essential issue that hinders the recent improvements on STG forecasting. Specifically, public datasets in this area usually possess only a few months of data, restricting the number of training instances that can be constructed. For example, the frequently used benchmarks PEMS-04 and PEMS-08 [7] only have around 17,000 instances in total, which is relatively limited compared to the datasets in other domains, such as images and text. Consequently, the learned models may overfit the training data, leading to inferior generalization performance.

Meanwhile, self-supervised learning has demonstrated great promise in a series of tasks on graphs. It derives supervisory signals from the data itself, usually exploiting the underlying structure of the data. Most of the best performing self-supervised methods are based on *contrastive learning* [10, 24, 29, 34]. As depicted in Figure 1(b), their basic idea is to maximize the agreement between representations of nodes or graphs with similar semantics (i.e., positive pairs), while minimizing those with unrelated semantic information (i.e., negative pairs). Generally, positive pairs are established by applying data augmentations to generate two views of the same input (termed *anchor*) [34, 43], and the negative pairs are formed between the anchor and *all* other inputs' views within a batch. Moreover, there are two schemes to utilize the contrastive learning signals, i.e., pretraining & fine-tuning [12, 24] or joint learning [35, 37].

Contribution. In light of the success of contrastive learning, we present the first systematic study to answer a key question: *can we leverage the additional self-supervision signals derived from contrastive learning techniques to alleviate data scarcity, so as to benefit STG forecasting?* If so, how should we integrate contrastive learning into spatio-temporal neural networks? Concretely, we identify the following questions to address.

Q1: What is the appropriate training scheme when integrating contrastive learning with STG forecasting?
Q2: For STG, there are two possible objects (node or graph) to conduct contrastive learning. Which one should we select?
Q3: How should we perform data augmentation to generate a positive pair? Does the choice of augmentation methods matter?
Q4: Given an anchor, should all other objects be considered as negatives? If not, how should we filter out unsuitable negatives?

In this paper, we give an affirmative answer by presenting a novel framework, entitled *STGCL*, which incorporates the merits of contrastive learning into spatio-temporal neural networks. The framework can be regarded as a handy training strategy to improve performance without extra computational cost during inference. We also provide explanations and insights that derive from the empirical findings. The above questions are answered as follows.

A1: We demonstrate the effectiveness of jointly conducting the forecasting and contrastive tasks over the pretraining scheme through evaluations on popular STG benchmarks.
A2: We explore feasible and efficient designs of contrastive learning on two separate objects – node and graph level. The empirical study shows that integrating graph-level contrast with the joint learning scheme achieves the best performance.
A3: We introduce four types of data augmentation methods for STG data which perturb the inputs in three aspects – graph structure, time domain, and frequency domain. Our experiments reveal that the model is not sensitive to the semantics of the proposed augmentations.
A4: We propose to filter out the hardest negatives per anchor based on the unique characteristics of STG, e.g., temporal closeness[1] and periodicity, yielding improvements on the prediction accuracy over real-world benchmarks.

2 PRELIMINARIES

2.1 Spatio-Temporal Graph Forecasting

We define a sensor network as a graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{A})$, where $\mathcal{V}$ is a set of nodes and $|\mathcal{V}| = N$, $\mathcal{E}$ is a set of edges, and $\mathbf{A} \in \mathbb{R}^{N\times N}$ is a weighted adjacency matrix where the edge weights represent the proximity between nodes. The graph G has a unique feature matrix $\mathbf{X}^t \in \mathbb{R}^{N\times F}$ at each time step t, where F is the feature dimension. The features consist of a target attribute (e.g., traffic speed or flow) and other auxiliary information, such as time of day [32]. In STG forecasting, we aim to learn a function f to predict the target attribute of the next T steps based on S historical frames:

$$\mathcal{G} : [\mathbf{X}^{(t-S):t}; G] \xrightarrow{f} \mathbf{Y}^{t:(t+T)} \tag{1}$$

where $\mathbf{X}^{(t-S):t} \in \mathbb{R}^{S\times N\times F}$ indicates the observations from the time step $t-S$ to t (excluding the right endpoint), and $\mathbf{Y}^{t:(t+T)} \in \mathbb{R}^{T\times N\times 1}$ denotes the T-step ahead predictions.

2.2 Contrastive Learning on General Graphs

Recent studies have verified the power of contrastive learning in learning unsupervised representations of graph data [34, 43]. Generally, the contrastive object varies with the downstream tasks, e.g., node-level contrastive learning is applied to learn useful representations before performing node classification tasks [43].

The pipeline of node- and graph-level contrastive learning can be summarized into a unified form (as illustrated in Figure 1(b)). Firstly, given an input graph, we utilize data augmentation methods to generate two correlated views. These views then pass through a GNN encoder to obtain the corresponding node or graph (derived from a readout function) representations. A projection head, i.e., non-linear transformations, further maps the representations to another latent space, where the contrastive loss is computed.

During training, a batch of M nodes/graphs are sampled and processed through the above procedure, resulting in $2M$ representations in total. Let $\mathbf{z}'_i$ and $\mathbf{z}''_i$ denote the latent features (i.e., the outputs of the projection head) of the first and second views of the ith node/graph, and $\text{sim}(\mathbf{z}'_i, \mathbf{z}''_i)$ denote the cosine similarity between them. Below shows an example of the contrastive loss applied in GraphCL [34], which is a variant of the InfoNCE loss [5, 22] and computed across the nodes/graphs in a batch:

$$\mathcal{L}_{cl} = \frac{1}{M}\sum_{i=1}^{M} -\log\frac{\exp(\text{sim}(\mathbf{z}'_i, \mathbf{z}''_i)/\tau)}{\sum_{j=1, j\neq i}^{M}\exp(\text{sim}(\mathbf{z}'_i, \mathbf{z}''_j)/\tau)} \tag{2}$$

where τ denotes the temperature parameter and a total of $M-1$ negatives are incorporated for the ith node/graph. After pretraining, we leverage the learned representations to perform downstream tasks, e.g, node or graph classification.

2.3 STG Encoder & Decoder

Based on the techniques exploited in the temporal dimension, existing STG forecasting models mainly fall into two classes: CNN-based and RNN-based methods. Here, we briefly introduce their architectures, which are generally in a encoder-decoder fashion.

For the form of the STG encoder $f_\theta(\cdot)$, CNN-based methods usually apply temporal convolutions and graph convolutions sequentially [32] or synchronously [27] to capture spatio-temporal dependencies. The widely applied form of temporal convolution is the dilated causal convolution [36], which enjoys an exponential growth of the receptive field by increasing the layer depth. After several layers of convolution, the temporal dimension is eliminated because the knowledge has been encoded into the representations. While in RNN-based methods, graph convolution is integrated with recurrent neural networks [18]. The hidden representation from the last step is usually seen as a summary of the encoder. In short, both CNN- and RNN-based models' encoder output

$$\mathbf{H} = f_\theta([\mathbf{X}^{(t-S):t}; G]) \tag{3}$$

[1]Temporal closeness means time intervals in the recent time [7, 20, 40]

where $\mathbf{H} \in \mathbb{R}^{N \times D}$, and D is the size of the hidden dimension.

For the form of the STG decoder $g_\phi(\cdot)$, CNN-based models often apply several linear layers to map high dimensional representations to low dimensional outputs [32]. In RNN-based models, they either employ a feed-forward network [1] or a recurrent neural network [18] for generating the forecasting results. Lastly, a prediction loss $\mathcal{L}_{pred}$, e.g., mean absolute error (MAE), is utilized to train the neural networks. Formally, we have

$$\mathcal{L}_{pred} = |\hat{\mathbf{Y}}^{t:(t+T)} - \mathbf{Y}^{t:(t+T)}|, \text{ where } \hat{\mathbf{Y}}^{t:(t+T)} = g_\phi(\mathbf{H}). \quad (4)$$

3 METHODOLOGY

In this section, we will introduce the proposed STGCL framework by addressing the four essential questions. We start by elaborating two possible schemes to incorporate contrastive learning with STG forecasting in Section 3.1. We then describe two designs of contrastive tasks on different levels (node and graph level) in Section 3.2. Afterwards, we introduce four kinds of data augmentations for STG data in Section 3.3 and illustrate a rule-based strategy to exclude undesirable negatives when calculating contrastive loss in Section 3.4. Finally, we link all the proposed techniques and provide a sample implementation in Section 3.5.

3.1 Training Schemes (Q1)

In this part, we discuss two candidate schemes to incorporate contrastive learning with STG forecasting.

3.1.1 Pretraining & fine-tuning. Following previous works [34, 43], we first generate the first view $\mathcal{G}'$ and the second view $\mathcal{G}''$ of the same input $\mathcal{G}$ by augmentation (see details in Section 3.3). Then, we feed the views through an STG encoder $f_\theta(\cdot)$ and a projection head $q_\psi(\cdot)$, which are trained with the contrastive objective $\mathcal{L}_{cl}$. Finally, we discard the projection head and fine-tune the encoder with an untrained decoder $g_\phi(\cdot)$ to predict the future via optimizing $\mathcal{L}_{pred}$.

3.1.2 Joint learning. In the joint learning approach, we need to simultaneously perform forecasting and contrastive tasks. However, using the original input for forecasting and two augmented views for contrasting induces unnecessary overhead. Instead, we use the original input $\mathcal{G}$ not only to conduct the forecasting task but also as the first view ($\mathcal{G}' = \mathcal{G}$), and only use augmentations to form the second view $\mathcal{G}''$ for contrastive learning.

Subsequently, an STG encoder $f_\theta(\cdot)$ is jointly trained with a projection head $q_\psi(\cdot)$ and an STG decoder $g_\phi(\cdot)$. In this case, the contrastive objective serves as an auxiliary regularization term during training and provides additional self-supervision signals for improving generalization [35]. The final form of the loss function is shown below, where λ is a trade-off parameter.

$$\mathcal{L} = \mathcal{L}_{pred} + \lambda \mathcal{L}_{cl} \quad (5)$$

3.2 What & How to Contrast (Q2)

As described in Section 2.3, the high-level representation extracted from an STG encoder is $\mathbf{H} \in \mathbb{R}^{N \times D}$, which means that each node has its latent features. Hence, we propose two feasible designs of contrastive learning methods, which contrast on different objects, i.e., node- or graph-level representations. The rationales behind the two designs are different. For node-level contrast, it is more fine-grained and matches to the level of the forecasting tasks, i.e., generating predictions at each node. In graph-level contrast, it considers global knowledge of the whole graph, which may aid each node for learning a more useful representation.

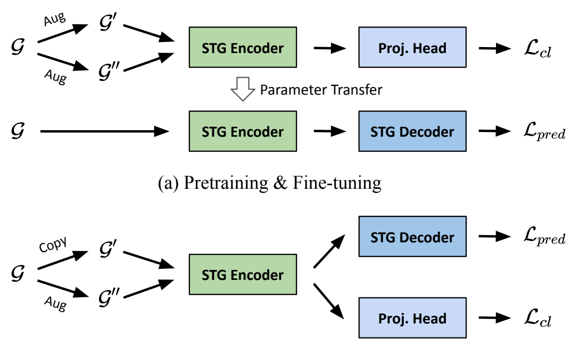


Figure 2: Illustration of different training schemes.

3.2.1 Node-level contrast. In node-level method, we directly take $\mathbf{H}$ and use a projection network $q_\psi(\cdot)$ to map it to another latent space $\mathbf{Z} \in \mathbb{R}^{N \times D}$. Suppose we have a batch of M STG with N nodes, given an anchor (a node representation), its positive comes from its augmented view. To perform full spatio-temporal contrast, its candidate negatives include all the other nodes in this time step and all the nodes in other time steps in this batch, resulting in MN negatives in total. When computing pair-wise cosine similarity, this method incurs high computational costs and memory usage (the computational complexity is $O(M^2N^2)$), which may become totally unaffordable in large graphs.

Inspired by existing arts [1, 32] that capture spatio-temporal dependencies separately, we address this issue by factorizing the full spatio-temporal contrast along the spatial and temporal dimensions. The negatives now come from the other nodes at this time step and the same node at the other time steps (see Figure 3). In this way, we only have $M + N$ negatives and the complexity decreases to $O(M^2 + N^2)$, leading to better efficiency and memory utility.

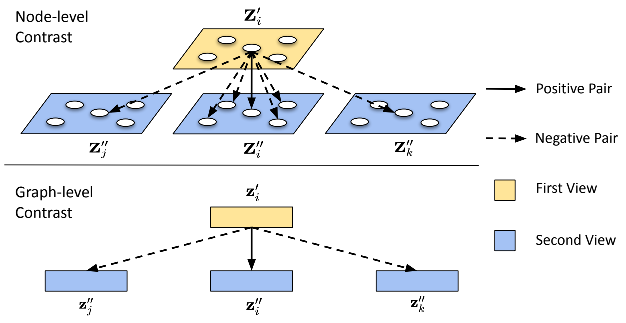


Figure 3: Sketch of node- and graph-level contrast. The index i denotes the anchor and indices j, k denote two arbitrary STG instances in this batch.

3.2.2 Graph-level contrast. In this approach, we first utilize a summation function as a readout function to obtain graph-level representation $\mathbf{s} \in \mathbb{R}^D$. This representation can be seen as a summary of a whole STG input. Then we apply a projection head $q_\psi(\cdot)$ to map it to the latent space $\mathbf{z} \in \mathbb{R}^D$. Given an anchor (a graph representation), its positive comes from its augmented version. Its potential negatives are the representations from the other graphs within this batch (see Figure 3). By contrasting at this level, a model is encouraged to distinguish different inputs' spatio-temporal patterns.

3.3 Data Augmentation (Q3)

Data augmentation is a necessary component of the contrastive learning techniques. It helps to build semantically similar pairs and assists the model to learn invariant representations under different types and levels of perturbations [5]. However, so far data augmentations have been less explored for STG. For example, the intrinsic properties of STG data (especially temporal dependencies) are not well considered in current graph augmentation methods [34, 37].

In this work, we modify two popular graph augmentations (edge perturbation [34] and attribute masking [34]) and propose two new approaches specifically designed for STG. The four methods perturb data in terms of graph structure, time domain, and frequency domain, thus making the learned representation less sensitive to changes of graph structure or signals. We denote $\mathbf{X}^{(t-S):t} \in \mathbb{R}^{S \times N}$ in this section, because only the target attribute (e.g., traffic speed) is modified. The details of each method are given as follows.

3.3.1 Edge masking. Edge perturbation [34] suggests either adding or deleting a certain ratio of edges to/from an unweighted graph. However, it is an awkward fit for the weighted adjacency matrix used in STG forecasting, since assigning proper weights to the added edges is difficult. Therefore, we make a revision by masking (deleting) entries of the adjacency matrix to perturb the graph structure. Each entry of the augmented matrix $\mathbf{A}'$ is given by:

$$\mathbf{A}'_{ij} = \begin{cases} \mathbf{A}_{ij}, & \text{if } \mathbf{M}_{ij} \geqslant r_{em} \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

where $\mathbf{M} \sim U(0,1)$ is a random matrix and r_{em} is tunable. We share the augmented matrix across the inputs within a batch for efficiency. This method is applicable for both predefined and adaptive adjacency matrix [32].

3.3.2 Input masking. We adjust the attribute masking [34] method by restricting the mask only on the target attribute. The goal is to strengthen the model's robustness to the factor of missing values, which is a general case in STG data. Hence, each entry of the augmented feature matrix $\mathbf{P}^{(t-S):t}$ is generated by:

$$\mathbf{P}^{(t-S):t}_{ij} = \begin{cases} \mathbf{X}^{(t-S):t}_{ij}, & \text{if } \mathbf{M}_{ij} \geqslant r_{im} \\ -1, & \text{otherwise} \end{cases} \tag{7}$$

where $\mathbf{M} \sim U(0,1)$ is a random matrix and r_{im} is tunable.

3.3.3 Temporal shifting. STG data derive from nature and evolve over time continuously. However, they can only be recorded by sensors in a discrete manner, e.g., a reading every five minutes. Motivated by this, we shift the data along the time axis to exploit the intermediate status between two consecutive time steps (see Figure 4). We implement this idea by linearly interpolating between consecutive inputs. Formally,

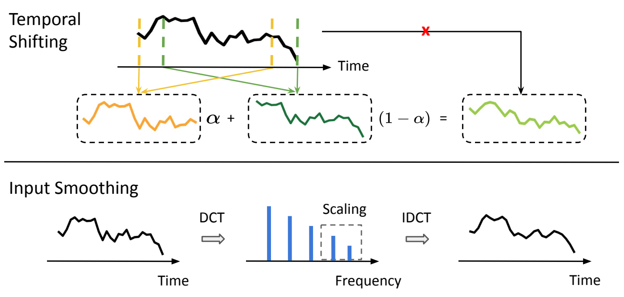


Figure 4: Sketch of temporal shifting and input smoothing.

$$\mathbf{P}^{(t-S):t} = \alpha \mathbf{X}^{(t-S):t} + (1-\alpha)\mathbf{X}^{(t-S+1):(t+1)} \tag{8}$$

where α is generated within the distribution $U(r_{ts}, 1)$ every epoch and r_{ts} is adjustable. This method is input-specific, which means different inputs have their unique α. Meanwhile, our operation can be related to the mixup augmentation [38]. The major difference is that we conduct weighted averaging between two successive time steps to ensure interpolation accuracy.

3.3.4 Input smoothing. To reduce the impact of data noise in STG, this method smooths the inputs by scaling high-frequency entries in the frequency domain (see Figure 4). Specifically, we first concatenate histories with future values (both are available during training) to enlarge the length of the time series to $L = S + T$, and obtain $\mathbf{X}^{(t-S):(t+T)} \in \mathbb{R}^{L \times N}$. Then we apply Discrete Cosine Transform (DCT) to convert the sequence of each node from the time domain to the frequency domain. We keep the low frequency E_{is} entries unchanged and scale the high frequency $L - E_{is}$ entries by the following steps: 1) We generate a random matrix $\mathbf{M} \in \mathbb{R}^{(L-E_{is}) \times N}$, which satisfies $\mathbf{M} \sim U(r_{is}, 1)$ and r_{is} is adjustable. 2) We leverage normalized adjacency matrix $\tilde{\mathbf{A}}$ to smooth the generated matrix by $\mathbf{M} = \mathbf{M}\tilde{\mathbf{A}}^2$. The intuition is that neighboring sensors should have similar scaling ranges and multiplying two steps of $\tilde{\mathbf{A}}$ should be sufficient to smooth the data. This step can be omitted when the adjacency matrix is not available. 3) We element-wise multiply the random numbers with the original $L - E_{is}$ entries. Finally, we use Inverse DCT (IDCT) to convert data back to the time domain.

3.4 Negative Filtering (Q4)

In most contrastive learning methods, all the other objects within a batch are seen as negatives for a given anchor. This ignores the fact that some of these objects may be semantically similar to the anchor, and are inappropriate to be used as negatives. There are some solutions suggesting using instance's label to assign positive and negative pairs [9, 14]. However, STG forecasting is a typical regression task without semantic labels. Thus, we propose to filter out unsuitable negatives based on the unique properties of STG data. Denoting χ_i as the set of acceptable negatives for the ith object after filtering, we extend the contrastive loss in Eq. 2 to:

$$\mathcal{L}_{cl} = \frac{1}{M}\sum_{i=1}^{M} -\log \frac{\exp(\mathrm{sim}(\mathbf{z}'_i, \mathbf{z}''_i)/\tau)}{\sum_{j\in\chi_i}\exp(\mathrm{sim}(\mathbf{z}'_i, \mathbf{z}''_j)/\tau)} \tag{9}$$

3.4.1 Spatial negative filtering. In the spatial dimension, we utilize the information from the predefined adjacency matrix. Specifically, the first-order neighbors of each node are excluded during contrastive loss computation. Note that this part is only applicable for node-level contrast.

3.4.2 Temporal negative filtering. Unlike the former method, temporal negative filtering can be applied to both levels. Here, we propose to filter negatives by utilizing the ubiquitous temporal properties of STG data – closeness and periodicity [40]. To facilitate understanding, Figure 5 depicts a screenshot of the traffic speed in a time period on the BAY dataset [18]. We can observe that the pattern from 6 am to 7 am on Monday is similar to that day from 7 am to 8 am (closeness), and also similar to the same time on Tuesday (daily periodic) and next Monday (weekly periodic). Hence, temporally close inputs (regardless of the day) are likely to have similar spatio-temporal representations in the latent space. If we form negative pairs by using these semantically similar inputs (i.e., hard negatives) and push apart their representations, it may break the semantic structure and worsen performance.

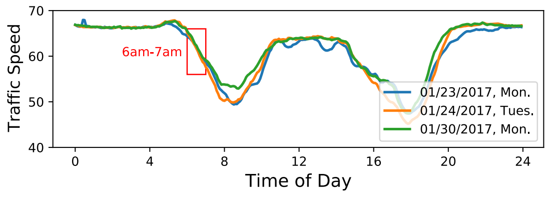


Figure 5: Example of temporal correlations in STG data.

Therefore, we devise a rule-based strategy that leverages the attribute "time of day" in the input to filter out the hardest negatives (i.e., the most similar inputs in semantics) per anchor. In this forecasting task, due to the lack of semantic labels, we define the hardest negatives using a controllable threshold r_f. Specifically, denoting the starting "time of day" of an anchor as t_i, we obtain the acceptable negatives within a batch by letting each input's starting "time of day" t satisfy the requirement of $|t - t_i| > r_f$. If r_f is set to zero, all the other objects in a batch are used to form the negative pairs with the anchor. We empirically find that the choice of r_f is of great importance to the quality of learned representations. For instance, using a very large r_f will significantly reduce the number of negatives, which may impair the contrastive task. For more details, please see Section 4.4.2.

3.5 Implementation

To ease understanding, here, we give a sample implementation to link all the techniques introduced above. Figure 6 shows the implementation of choosing joint learning scheme and contrasting at graph level. Specifically, we first utilize an STG encoder to map both the original input and the augmented input to high-level representations $\mathbf{H}', \mathbf{H}''$. Then, the representations flow into the following two branches.

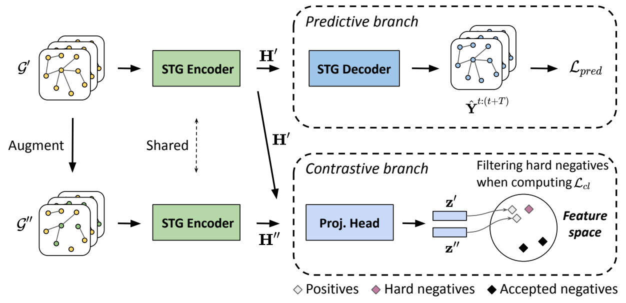


Figure 6: Sample implementation of STGCL. The missing edges and green nodes in $\mathcal{G}''$ indicate the perturbations on graph structure and signals, respectively.

- A *predictive branch* that feeds the representation $\mathbf{H}'$ through an STG decoder to forecast the future steps. The predictions of the decoder $\hat{\mathbf{Y}}^{t:(t+T)}$ are used to compute the forecasting loss with the ground truth.
- A *contrastive branch* that takes both $\mathbf{H}'$ and $\mathbf{H}''$ as inputs to conduct the auxiliary contrastive task. Specifically, we utilize a readout function to obtain the spatio-temporal summaries $\mathbf{s}', \mathbf{s}'' \in \mathbb{R}^D$ of the input. We further map the summaries to the latent space $\mathbf{z}', \mathbf{z}'' \in \mathbb{R}^D$ by a projection head. The applied projection network has two linear layers, where the first layer is followed by batch normalization and rectified linear units. Afterwards, a contrastive loss is used to maximize the similarities between $\mathbf{z}'$ and $\mathbf{z}''$, and minimize the similarities between $\mathbf{z}'$ and other inputs' augmented view. Lastly, we use negative filtering to avoid forming negative pairs between the most semantically similar inputs.

In addition, Algorithm 1 summarizes the training algorithm of the implementation. Line 6 is the code for predictive branch. Line 8-10 include the code for contrastive branch. Note that the data augmentation module aug() comprises the four proposed augmentation methods. The methods can be applied either solely or jointly.

4 EXPERIMENTS

In this section, we broadly evaluate, investigate, and interpret the benefits of integrating contrastive learning into STG forecasting. We first detail the experimental setup in Section 4.1. Then, we present the empirical results of using different training schemes and contrastive learning methods in Section 4.2 and 4.3, respectively. Finally, we provide an ablation study on the effects of data augmentation and negative filtering in Section 4.4.

4.1 Experimental Setup

4.1.1 Datasets. We assess STGCL on two popular traffic benchmarks from literature: PEMS-04 and PEMS-08 [7]. To have a fair comparison to prior works, an additional evaluation dataset is included in Table 5. There are three kinds of traffic measurements contained in the raw data, including traffic flow, average speed, and average occupancy. These traffic readings are aggregated into 5-minute windows, resulting in 288 data points per day. Following common practice [7], we use traffic flow as the target attribute and use the 12-step historical data to predict the next 12 steps. Z-score

Algorithm 1: Training algorithm of the implementation.

```
Input: Training data $\mathcal{G}$, labels Y, constant $r_f$, $\tau$, $\lambda$,
       STG encoder $f_\theta(\cdot)$ and decoder $g_\phi(\cdot)$,
       projection head $q_\psi(\cdot)$,
       augmentation module aug(),
       readout function readout(),
       negative filtering operation filter().
Output: $f_\theta(\cdot)$ and $g_\phi(\cdot)$.
1  for sampled batch $\{\mathcal{G}_i\}_{i=1}^M \in \mathcal{G}$ do
2      for $i = 1$ to $M$ do
3          $\mathcal{G}_i' = \mathcal{G}_i$;
4          $\mathcal{G}_i'' = \text{aug}(\mathcal{G}_i)$;
5          $\mathbf{H}_i', \mathbf{H}_i'' = f_\theta(\mathcal{G}_i'), f_\theta(\mathcal{G}_i'')$;
6          # predictive branch;
7          $\hat{\mathbf{Y}}_i = g_\phi(\mathbf{H}_i')$;
8          # contrastive branch;
9          $\mathbf{s}_i', \mathbf{s}_i'' = \text{readout}(\mathbf{H}_i'), \text{readout}(\mathbf{H}_i'')$;
10         $\mathbf{z}_i', \mathbf{z}_i'' = q_\psi(\mathbf{s}_i'), q_\psi(\mathbf{s}_i'')$;
11         $\chi_i \leftarrow \text{filter}(\mathcal{G}_i, r_f)$;
12     end
13     $\mathcal{L}_{pred} = \frac{1}{M}\sum_{i=1}^{M} |\hat{\mathbf{Y}}_i - \mathbf{Y}_i|$;
14     $\mathcal{L}_{cl} = \frac{1}{M}\sum_{i=1}^{M} -\log \frac{\exp(\text{sim}(\mathbf{z}_i', \mathbf{z}_i'')/\tau)}{\sum_{j \in \chi_i} \exp(\text{sim}(\mathbf{z}_i', \mathbf{z}_j'')/\tau)}$;
15     update $f_\theta(\cdot)$, $g_\phi(\cdot)$, and $q_\psi(\cdot)$ to minimize $\mathcal{L}_{pred} + \lambda\mathcal{L}_{cl}$;
16 end
17 return $f_\theta(\cdot)$ and $g_\phi(\cdot)$;
```

normalization is applied to the input data for fast training. The adjacency matrix is constructed by road-network distance with the thresholded Gaussian kernel method [26]. The datasets are divided into three parts for training, validation, and testing with a ratio of 6:2:2. More details are provided in Table 1 and Appendix A.1.

Table 1: Dataset statistics.

Datasets	#Nodes	#Edges	#Instances	Interval
PEMS-04	307	209	16,992	5 min
PEMS-08	170	137	17,856	5 min

4.1.2 Base Models. According to the performance comparison presented at LibCity[2] and [16], we select four models that have the top performance. They belong to the following two classes: CNN-based models (GWN, MTGNN) and RNN-based models (DCRNN, AGCRN).

- GWN: Graph WaveNet combines adaptive adjacency matrix with graph convolution and uses dilated casual convolution [32].
- MTGNN: This approach proposes a graph learning module and applies graph convolution with mix-hop propagation and dilated inception layer [31].
- DCRNN: Diffusion Convolution Recurrent Neural Network, which integrates diffusion convolution into RNNs [18].
- AGCRN: This approach develops two adaptive modules to enhance graph convolution and combines them into RNNs [1].

[2]https://libcity.ai/

Table 2: Test MAE results of the average values over all predicted time steps when integrating two contrastive tasks (node and graph) into two base models (GWN and AGCRN) through two training schemes: pretraining & fine-tuning (P&F) and joint learning (JL). We highlight the best performance in bold font.

Methods	PEMS-04	PEMS-08
GWN	19.33±0.11	14.78±0.03
w/ P&F-node	20.22±0.22	15.37±0.08
w/ P&F-graph	20.67±0.13	15.86±0.15
w/ JL-node	18.89±0.05	14.63±0.07
w/ JL-graph	**18.88±0.04**	**14.61±0.03**
AGCRN	19.39±0.03	15.79±0.06
w/ P&F-node	19.70±0.07	17.21±0.14
w/ P&F-graph	20.39±0.10	17.92±0.10
w/ JL-node	19.32±0.06	15.78±0.09
w/ JL-graph	**19.13±0.05**	**15.62±0.07**

4.1.3 Experiment Details. We use PyTorch 1.8 to implement all the base models and our method. We basically employ the default settings of base models from their source code, such as model configurations, batch size, optimizer, learning rate, and gradient clipping. The model-related settings of STGCL follow the same as in base models' implementation, except that we don't apply weight decay, as this will introduce another term in the loss function. For STGCL's specific settings, the temperature τ is set to 0.1 and the usage of augmentations and r_f are described in Section 4.4. All experiments are repeated five times with different seeds. We adopt three common metrics in forecasting tasks to evaluate the model performance, including mean absolute error (MAE), root mean squared error (RMSE), and mean absolute percentage error (MAPE). See more details about the settings in Appendix A.2 and A.3.

4.2 Training Schemes Evaluation (Q1)

We first examine the two training schemes mentioned in Q1, i.e., pretraining & fine-tuning and joint learning, to integrate contrastive learning with STG forecasting. Table 2 summarizes the experimental results on two datasets by using two base models (GWN and AGCRN) and two contrastive learning tasks (node and graph level). The results demonstrate the effectiveness of collectively performing the forecasting and contrastive learning tasks, no matter what the models and contrastive objects are applied. This fact indicates that joint learning is the preferable way to provide additional self-supervision signals. We also plot the learning curves of the two losses ($\mathcal{L}_{cl}$ and $\mathcal{L}_{pred}$) in Figure 7 (left hand side) to verify the learning process of joint learning. The reason why the contrastive loss is less than zero is that the denominator of the contrastive loss does not contain the numerator, thus their ratio can be greater than one. While on the right hand side of the figure, we show the trends of the validation loss. It is clear that GWN w/ JL-graph has better performance and is more stable than the pure GWN model.

Then, we investigate *whether pretraining by contrastive learning can really benefit STG forecasting*. Specifically, we use a smaller learning rate during fine-tuning, i.e., 10% of the learning rate applied in the base model, with the expectation to retain the information

Table 3: Test MAE results of three specific horizons when incorporating two contrastive tasks into four state-of-the-art models through joint learning (JL). Δ denotes the sum of improvements against the base models for the three horizons. min: minutes.

Methods	PEMS-04				PEMS-08			
	15 min	30 min	60 min	Δ	15 min	30 min	60 min	Δ
GWN	18.20±0.09	19.32±0.13	21.10±0.18	–	13.80±0.05	14.75±0.04	16.39±0.09	–
w/ JL-node	17.97±0.05	18.90±0.07	**20.38±0.07**	-1.37	13.67±0.06	14.63±0.08	16.14±0.13	-0.50
w/ JL-graph	**17.93±0.04**	**18.87±0.04**	20.40±0.09	**-1.42**	**13.67±0.04**	**14.61±0.03**	**16.09±0.05**	**-0.57**
MTGNN	18.32±0.05	19.10±0.05	20.39±0.09	–	14.36±0.06	15.34±0.10	16.91±0.16	–
w/ JL-node	18.03±0.02	18.79±0.06	19.94±0.03	-1.05	14.05±0.05	14.94±0.04	16.38±0.09	-1.24
w/ JL-graph	**17.99±0.03**	**18.72±0.05**	**19.88±0.07**	**-1.22**	**14.04±0.05**	**14.90±0.05**	**16.23±0.08**	**-1.44**
DCRNN	19.99±0.11	22.40±0.19	27.15±0.35	–	15.23±0.15	16.98±0.25	20.27±0.41	–
w/ JL-node	19.94±0.08	22.38±0.14	27.15±0.26	-0.07	**15.15±0.05**	**16.85±0.11**	**20.02±0.23**	**-0.46**
w/ JL-graph	**19.82±0.08**	**22.07±0.12**	**26.51±0.21**	**-1.14**	15.19±0.11	16.89±0.19	20.09±0.34	-0.31
AGCRN	18.53±0.03	19.43±0.06	20.72±0.03	–	14.58±0.07	15.71±0.07	17.82±0.11	–
w/ JL-node	18.46±0.04	19.37±0.07	20.69±0.11	-0.16	14.55±0.06	15.69±0.10	17.82±0.14	-0.05
w/ JL-graph	**18.31±0.04**	**19.17±0.06**	**20.39±0.03**	**-0.81**	**14.51±0.05**	**15.56±0.06**	**17.51±0.10**	**-0.53**

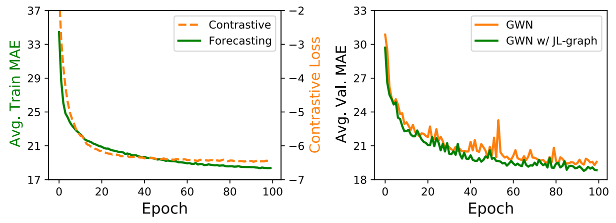


Figure 7: The left part shows the training curves of forecasting loss and contrastive loss by using GWN w/ JL-graph on PEMS-04. The right part shows the validation loss of GWN w/ JL-graph against pure GWN on PEMS-04.

that has learned during pretraining. From Table 2, it can be seen that pretraining & fine-tuning performs worse than the base model, revealing the inefficacy of pretraining by contrastive learning. Going into deeper, according to a recent work [30], contrastive learning can be viewed as a direct optimization of two properties – *alignment* and *uniformity*. The uniformity enforces the representations to be uniformly distributed on the unit hypersphere, thus facilitating different classes to be well clustered and linearly separable. This distribution provides more benefits to a classification problem, but may not capture useful information for the forecasting.

Additionally, in the pretraining & fine-tuning setting, we observe that the node-level contrast method outperforms the graph-level contrast. The results indicate that we have to ensure the consistency between the pretraining task and the downstream task, i.e., the prior knowledge extracted from the graph-level contrast cannot help the forecasting task that is performed on the node level. On the contrary, we find graph-level contrast surpasses node-level contrast in the joint learning setting, which will be detailed in Section 4.3.

4.3 Node & Graph-Level Contrast (Q2)

We have demonstrated the power of the joint learning scheme in improving STG forecasting. In this section, we explore how the two levels of contrastive tasks behave on four state-of-the-art models.

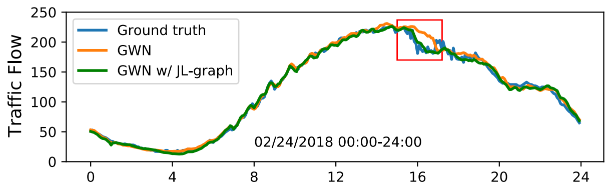


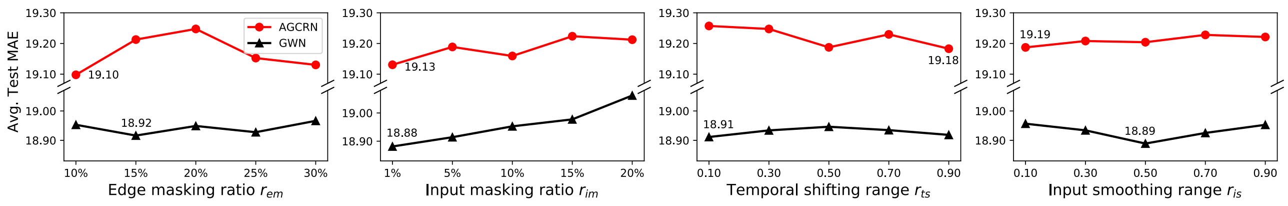
Figure 8: Visualization of 60 minutes-ahead predictions on a snapshot of the PEMS-04 test set. x-axis: the time of day.

Table 3 presents the MAE results on two datasets. See the complete table (including MAE, RMSE, MAPE) in Table 5.

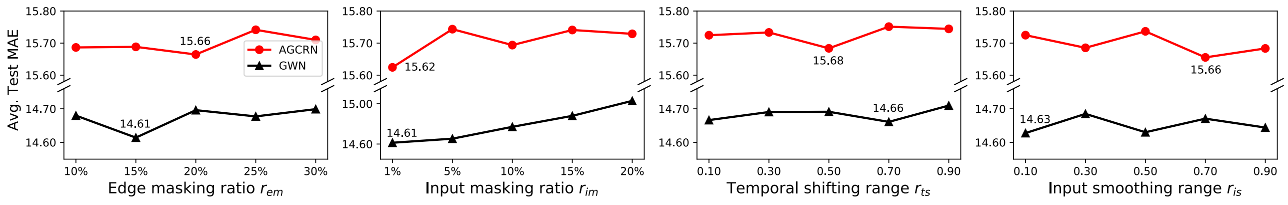
Our results show the effectiveness of utilizing contrastive loss over all base models on all datasets. In particular, the graph-level method consistently outperforms the base models, while the node-level method achieves improvements in most cases. According to the student t-test, 54%/83% of the improvements are statistically significant at level 0.05 for node/graph level, respectively. These findings suggest that the proposed contrastive methods are model-agnostic, i.e., they can be plugged into both CNN-based and RNN-based architectures. In addition, the standard deviation of contrastive methods is generally smaller than that of the base model, indicating the additional stability provided by them.

Moreover, we can see that in general, the graph-level contrast performs better than the node-level contrast. We speculate the reasons as follows. In the joint learning setting, nodes are guided to perform forecasting tasks, which results in a certain distribution in the latent space. Meanwhile, nodes are enforced to distinguish other nodes by the contrastive objective. In light of these factors, it is non-trivial to learn a powerful contrastive component, leading to minor improvements to the predictive performance. In contrast, it might be easier to distinguish patterns at the graph level. Also, since the number of neighbors to a certain node in these traffic datasets is usually small, it is more important and effective to utilize the graph-level contrast to provide global contexts to each node.

Another important observation in Table 3 is that contrastive methods achieve larger improvements for long-term predictions.



(a) PEMS-04



(b) PEMS-08

Figure 9: Effects of different augmentation methods on the PEMS-04 and PEMS-08 datasets.

To investigate it, we use GWN as the base model and randomly select a sensor (#200) from PEMS-04 for a case study. Figure 8 visualizes the 60 minutes-ahead prediction results against the ground truth on a snapshot of the test data. It can be seen that GWN w/ JL-graph outperforms the counterpart, especially at the sudden change (see the red rectangle). A plausible explanation is that model has learned discriminative representations by receiving signals from the contrastive task during training, hence it can successfully distinguish some distinct patterns (like sudden changes).

4.4 Ablation Studies (Q3 & Q4)

In this section, we set the joint learning scheme and the graph-level contrast as default due to their superior performance against their counterparts. Then we select one CNN-based model (GWN) and one RNN-based model (AGCRN) as representatives to study the effects of data augmentation and negative filtering.

4.4.1 Effects of data augmentation. We show the effects of different augmentation methods by tuning the hyper-parameter related to that method in Figure 9. We highlight the best performance achieved by each model on each augmentation method in the figure. For input smoothing, the fixed entries E_{is} is set to 20 after tuning within the range of {16, 18, 20, 22}.

From the figures, we have the following observations: 1) all the proposed data augmentations yield gains over the base models. 2) The best performance achieved by each augmentation method has little difference, though the semantics of each method is different (e.g., temporal shifting and input smoothing are performed in the time and frequency domain, respectively). This indicates that our framework is not sensitive to the semantics of the proposed augmentations. This finding is also different from the situation in the vision domain, where data augmentation can notably vary the semantics and has a significant impact on the performance of the contrastive methods. 3) Input masking is significantly affected by the perturbation magnitude, while other augmentation methods are relatively stable and have no obvious trend. The reason could be that masking a certain fraction of entries to zero perturbs input a lot, but the perturbations injected by other augmentations are considered reasonable by the models. 4) Input masking with a 1% ratio achieves most of the best performance across the applied models and datasets. We thereby suggest using this setting in other datasets for initiatives.

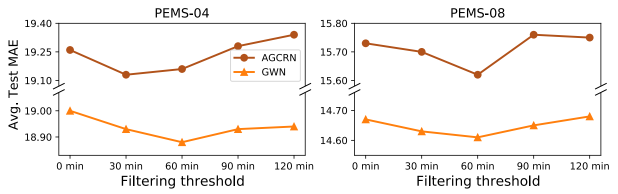


Figure 10: Effects of threshold r_f on PEMS-04 and PEMS-08.

4.4.2 Effects of negative filtering. We show the effects of the filtering threshold r_f in Figure 10. Input masking with a 1% ratio is used as the default augmentation setting. According to the results, we can see that the filtering threshold r_f works best when set to 30 or 60 minutes, which demonstrates the effectiveness of our simple solution. We also notice that when r_f is set to 120 minutes, some results are worse than when r_f is set to zero. The reason is that the threshold of 120 minutes filters out too many negatives, which eases the contrastive learning task. In this case, the model may not be able to learn useful discrimination knowledge that can benefit the forecasting task. To sum up, by filtering out the hardest negatives (i.e., the most semantically similar inputs), we enable the contrastive loss to focus on the *true* negatives. On the other hand, filtering too many negatives makes the contrastive task meaningless and leads to performance degradation.

Our negative filtering operation is also more efficient in measuring the similarity of two temporal sequences than existing methods such as dynamic time warping [3] or Pearson correlation coefficient. The reason is that our method only needs to compare the starting time of each input and does not involve other computations.

4.5 Result Summaries & Future Directions

We summarize and discuss the experimental findings in this section, answering the four questions mentioned in Section 1. Firstly, the joint learning scheme is able to consistently enhance base STG models, where the contrastive component works as a regularizer. Pretraining & fine-tuning scheme optimizes different objective functions at the two stages, i.e., from self-supervision to forecasting loss, and we show that the knowledge derived from pretrained contrastive tasks does not improve forecasting performance. This finding is similar to the observation in the graph domain [35].

Second, for the self-supervised tasks, we find that the node-level contrast outperforms the graph-level contrast in the pretraining scheme, but performs worse in the joint learning setting. This indicates that the auxiliary task cannot conflict with the main (forecasting) task in joint learning. In addition, since masked image modeling has surpassed the performance achieved by contrastive methods recently [2, 11], a future study may focus on acquiring additional training signals from masked modeling.

Third, we empirically show that the model is not sensitive to the semantics and the perturbation magnitude (only except for input masking) of the proposed augmentations. Motivated by the empirical results from SimCSE [6], a future study may conduct to apply a single dropout as the augmentation. Moreover, a possible direction for improving performance is to leverage adaptive data augmentation techniques [43] to identify the importance of nodes/edges, thus treating them differently when generating augmented views.

Last, we should carefully select the negatives, i.e., the negatives should not be too similar/dissimilar to the anchor point. Designing a filtering method that dynamically assigns weights to hard negatives rather in a heuristic-based way could be one future direction.

5 RELATED WORK

5.1 Deep Learning for STG Forecasting

STG forecasting is a typical problem in smart city efforts, facilitating a wide range of applications [42]. In recent years, deep neural networks have become the dominant class for modeling STG [8, 13, 25, 33, 41]. Generally, they integrate graph convolutions with either CNNs or RNNs to capture the spatial and temporal dependencies in STG. As a pioneering work, DCRNN [18] considers traffic flow as a diffusion process and proposes a novel diffusion convolution to capture spatial correlations. To improve training speed, GWN [32] adopts a complete convolutional structure, which combines graph convolution with dilated casual convolution operation. In addition, attention mechanisms [7, 19] are proposed to capture dynamic inter- and inner-sensor correlations. The recent developments in this field show the following trends: making the adjacency matrix fully learnable [1], and developing modules to jointly capture spatial and temporal dependencies [27]. However, these models are usually trained on datasets with limited instances, which may lead to overfitting problem and inferior generalization performance.

5.2 Graph-based Contrastive Learning

Recently, contrastive representation learning on graphs has attracted significant attention. According to a recent survey [21], existing efforts can be categorized into cross-scale contrasting and same-scale contrasting.

Cross-scale contrasting refers to the scenario that the contrastive elements are in different scales. A representative work named DGI [29] contrasts between the patch and graph-level representations via mutual information (MI) maximization, thus gaining benefits from flowing global information to local representations. While MVGRL [10] suggests a multi-view contrasting by using a diffused graph as a global view of the original graph, after which the MI is maximized in a cross-view and cross-scale manner.

For same-scale contrasting, the elements are in an equal scale, such as graph-graph contrasting. Regarding the definition of positive/negative pairs, the approaches can be further divided into context-based and augmentation-based. The context-based methods generally utilize random walks to obtain positives. Our work lies in the scope of augmentation-based methods, where the positives are generated by perturbations on nodes or edges. For example, GraphCL [34] adopts four augmentations to form positive pairs and contrasts at graph level. GCA [43] works at the node level and performs augmentations that are adaptive to the graph structure and attributes. IGSD [39] uses graph diffusion to generate augmented views and applies a teacher-student framework.

It can be seen that the current works mostly focus on developing contrastive methods for attributed graphs and pay less attention to more complex graphs, e.g., heterogeneous graphs and STG [21]. In this work, we present the first exploration on adapting contrastive learning to STG.

6 CONCLUSION

In this paper, we present the first systematic study on incorporating contrastive learning into STG forecasting. We first elaborate two potential schemes for integrating contrastive learning. Then we propose two feasible and efficient designs of contrastive tasks that are performed on the node or graph level. In addition, we introduce four types of STG-specific data augmentations to construct positive pairs and propose a rule-based strategy to alleviate an inherent shortcoming of the contrastive methods, i.e., ignoring input's semantic similarity. Our extensive evaluations on real-world STG benchmarks and different base models demonstrate that integrating graph-level contrast with the joint learning scheme achieves the best performance. We also provide rational explanations and insights for experimental results and discuss the future research directions.

7 ACKNOWLEDGEMENTS

This research is supported by Singapore Ministry of Education Academic Research Fund Tier 2 under MOE's official grant number T2EP20221-0023. We thank all reviewers for their constructive suggestions to improve this paper. We also thank Hengchang Hu, Zhiyuan Liu, and Canwei Liu for their valuable feedback.

REFERENCES

[1] Lei Bai, Lina Yao, Can Li, Xianzhi Wang, and Can Wang. 2020. Adaptive Graph Convolutional Recurrent Network for Traffic Forecasting. In *Proceedings of Advances in Neural Information Processing Systems*. 17804–17815.

[2] Hangbo Bao, Li Dong, Songhao Piao, and Furu Wei. 2022. Beit: Bert pre-training of image transformers. In *The 10th International Conference on Learning Representations*.

[3] Donald J Berndt and James Clifford. 1994. Using dynamic time warping to find patterns in time series. In *ACM KDD Workshop*. 359–370.

[4] Defu Cao, Yujing Wang, Juanyong Duan, Ce Zhang, Xia Zhu, Congrui Huang, Yunhai Tong, Bixiong Xu, Jing Bai, Jie Tong, et al. 2020. Spectral temporal graph neural network for multivariate time-series forecasting. In *Proceedings of Advances in Neural Information Processing Systems*. 17766–17778.

[5] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning*. 1597–1607.

[6] Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. Simcse: Simple contrastive learning of sentence embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 6894–6910.

[7] Shengnan Guo, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. 2019. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *AAAI*. 922–929.

[8] Liangzhe Han, Bowen Du, Leilei Sun, Yanjie Fu, Yisheng Lv, and Hui Xiong. 2021. Dynamic and Multi-faceted Spatio-temporal Deep Learning for Traffic Speed Forecasting. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 547–555.

[9] Tengda Han, Weidi Xie, and Andrew Zisserman. 2020. Self-supervised co-training for video representation learning. In *Proceedings of Advances in Neural Information Processing Systems*.

[10] Kaveh Hassani and Amir Hosein Khasahmadi. 2020. Contrastive multi-view representation learning on graphs. In *Proceedings of the 37th International Conference on Machine Learning*. 4116–4126.

[11] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. 2022. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 16000–16009.

[12] Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. 2020. Gpt-gnn: Generative pre-training of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1857–1867.

[13] Yingfan Huang, Huikun Bi, Zhaoxin Li, Tianlu Mao, and Zhaoqi Wang. 2019. Stgat: Modeling spatial-temporal interactions for human trajectory prediction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 6272–6281.

[14] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. 2020. Supervised contrastive learning. In *Proceedings of Advances in Neural Information Processing Systems*.

[15] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *The 5th International Conference on Learning Representations*.

[16] Fuxian Li, Jie Feng, Huan Yan, Guangyin Jin, Fan Yang, Funing Sun, Depeng Jin, and Yong Li. 2021. Dynamic graph convolutional recurrent network for traffic prediction: Benchmark and solution. *ACM Transactions on Knowledge Discovery from Data* (2021).

[17] Mengzhang Li and Zhanxing Zhu. 2021. Spatial-Temporal Fusion Graph Neural Networks for Traffic Flow Forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 4189–4196.

[18] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. 2018. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *The 6th International Conference on Learning Representations*.

[19] Yuxuan Liang, Songyu Ke, Junbo Zhang, Xiuwen Yi, and Yu Zheng. 2018. Geoman: Multi-level attention networks for geo-sensory time series prediction.. In *Proceedings of International Joint Conference on Artificial Intelligence*. 3428–3434.

[20] Yuxuan Liang, Kun Ouyang, Junkai Sun, Yiwei Wang, Junbo Zhang, Yu Zheng, David Rosenblum, and Roger Zimmermann. 2021. Fine-Grained Urban Flow Prediction. In *Proceedings of the Web Conference 2021*. 1833–1845.

[21] Yixin Liu, Shirui Pan, Ming Jin, Chuan Zhou, Feng Xia, and Philip S Yu. 2021. Graph self-supervised learning: A survey. *arXiv preprint arXiv:2103.00111* (2021).

[22] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748* (2018).

[23] Zheyi Pan, Yuxuan Liang, Weifeng Wang, Yong Yu, Yu Zheng, and Junbo Zhang. 2019. Urban traffic prediction from spatio-temporal data using deep meta learning. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1720–1730.

[24] Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. 2020. Gcc: Graph contrastive coding for graph neural network pre-training. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1150–1160.

[25] Hongzhi Shi, Quanming Yao, Qi Guo, Yaguang Li, Lingyu Zhang, Jieping Ye, Yong Li, and Yan Liu. 2020. Predicting origin-destination flow via multi-perspective graph convolutional network. In *Proceedings of the 36th International Conference on Data Engineering*. 1818–1821.

[26] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. 2013. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine* (2013), 83–98.

[27] Chao Song, Youfang Lin, Shengnan Guo, and Huaiyu Wan. 2020. Spatial-temporal synchronous graph convolutional networks: A new framework for spatial-temporal network data forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 914–921.

[28] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. In *The 6th International Conference on Learning Representations*.

[29] Petar Velickovic, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. 2019. Deep Graph Infomax. In *The 7th International Conference on Learning Representations*.

[30] Tongzhou Wang and Phillip Isola. 2020. Understanding contrastive representation learning through alignment and uniformity on the hypersphere. In *Proceedings of the 37th International Conference on Machine Learning*. 9929–9939.

[31] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, Xiaojun Chang, and Chengqi Zhang. 2020. Connecting the dots: Multivariate time series forecasting with graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 753–763.

[32] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. 2019. Graph wavenet for deep spatial-temporal graph modeling. In *Proceedings of International Joint Conference on Artificial Intelligence*. 1907–1913.

[33] Tong Xia, Junjie Lin, Yong Li, Jie Feng, Pan Hui, Funing Sun, Diansheng Guo, and Depeng Jin. 2021. 3DGCN: 3-Dimensional Dynamic Graph Convolutional Network for Citywide Crowd Flow Prediction. *ACM Transactions on Knowledge Discovery from Data* (2021), 1–21.

[34] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. 2020. Graph contrastive learning with augmentations. In *Proceedings of Advances in Neural Information Processing Systems*.

[35] Yuning You, Tianlong Chen, Zhangyang Wang, and Yang Shen. 2020. When does self-supervision help graph convolutional networks?. In *Proceedings of the 37th International Conference on Machine Learning*. 10871–10880.

[36] Fisher Yu and Vladlen Koltun. 2016. Multi-scale context aggregation by dilated convolutions. In *The 4th International Conference on Learning Representations*.

[37] Jiaqi Zeng and Pengtao Xie. 2021. Contrastive self-supervised learning for graph classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

[38] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. 2017. mixup: Beyond empirical risk minimization. In *The 5th International Conference on Learning Representations*.

[39] Hanlin Zhang, Shuai Lin, Weiyang Liu, Pan Zhou, Jian Tang, Xiaodan Liang, and Eric P Xing. 2021. Iterative graph self-distillation. In *The Workshop on Self-Supervised Learning for the Web*.

[40] Junbo Zhang, Yu Zheng, and Dekang Qi. 2017. Deep spatio-temporal residual networks for citywide crowd flows prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 1655–1661.

[41] Weijia Zhang, Hao Liu, Yanchi Liu, Jingbo Zhou, and Hui Xiong. 2020. Semi-supervised hierarchical recurrent graph neural network for city-wide parking availability prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 1186–1193.

[42] Yu Zheng, Licia Capra, Ouri Wolfson, and Hai Yang. 2014. Urban computing: concepts, methodologies, and applications. *ACM Transactions on Intelligent Systems and Technology* (2014), 1–55.

[43] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. 2021. Graph contrastive learning with adaptive augmentation. In *Proceedings of the Web Conference 2021*. 2069–2080.

A REPRODUCIBILITY

Here, we detail the experimental settings to support reproducibility of the reported results in the paper. First, we present the dataset-related settings in Section A.1. Second, we provide the specific settings in Section A.2 and A.3 that are used to obtain the results in Table 3 and 5. Third, Section A.4 shows the procedure of the pretraining & fine-tuning experiments.

A.1 Datasets

We conduct the experiments on the traffic datasets of PEMS-04[3], PEMS-08[3], and BAY[4].

- PEMS-04 [7]: The dataset refers to the traffic flow data in the Bay Area. There are 307 sensors and the period of data ranges from Jan. 1 - Feb. 28, 2018.
- PEMS-08 [7]: The dataset contains traffic flow information collected from 170 sensors in the San Bernardino area from Jul. 1 - Aug. 31, 2016.
- BAY [18]: This dataset contains traffic speed information from 325 sensors in the Bay Area. It has 6 months of data ranging from Jan. 1 - Jun. 30, 2017.

We use one-hour (12-step) historical data to predict the future one-hour (12-step) values. The "time of the day" is used as an auxiliary feature for the inputs. The datasets are split into three parts for training, validation, and testing with a ratio of 6:2:2 on PEMS-04 and PEMS-08, and 7:1:2 on BAY. The statistics of data partitions is in Table 4. Following Li et al. [18], we build the adjacency matrix of the sensor graph in each dataset by using pairwise road network distances with a thresholded Gaussian kernel [26].

$$W_{ij} = \begin{cases} \exp(-\frac{\text{dist}(v_i,v_j)^2}{\sigma^2}), & \text{if } \text{dist}(v_i,v_j) \leqslant k \\ 0, & \text{otherwise} \end{cases}$$

where W_{ij} is the edge weight between sensor v_i and sensor v_j, $\text{dist}(v_i, v_j)$ denotes the road network distance between them, σ is the standard deviation of distances, and k is a threshold, which we set to 0.1. We provide a visualization of sensor distribution as well as the corresponding adjacency matrix on BAY in Figure 11.

Table 4: Details of data partition.

Datasets	#Training	#Validation	#Testing
PEMS-04	10,172	3,375	3,376
PEMS-08	10,690	3,548	3,549
BAY	36,465	5,209	10,419

A.2 Settings of the Base Models

We apply four base models that belong to the classes of CNN-based (GWN, MTGNN) and RNN-based methods (DCRNN, AGCRN). We basically employ the default settings of base models from their public source code. The common settings of the four models: Adam optimizer is applied, batch size is 64, and epochs are 100. The other settings of each model are listed as follows.

[3]https://github.com/guoshnBJTU/ASTGCN-r-pytorch
[4]https://github.com/liyaguang/DCRNN

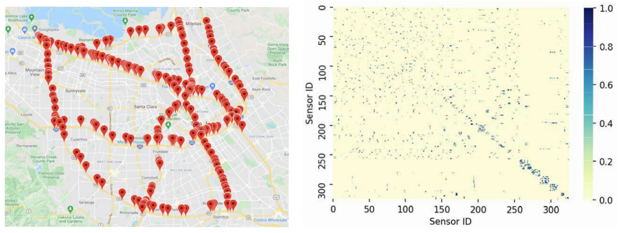


Figure 11: Visualizations of sensor distribution and the adjacency matrix on BAY.

- GWN [32]: In this model, a spatio-temporal layer is constructed by a gated temporal convolution layer and a graph convolution layer. To cover the input sequence, eight spatio-temporal layers are applied and the dilation factors of each layer are 1, 2, 1, 2, 1, 2, 1, 2. The diffusion step and dropout rate in each graph convolution layer are set to 2 and 0.3. The hidden dimension size is 32. The learning rate, weight decay, and gradient clipping threshold are set to $1e^{-3}$, $1e^{-4}$, and 5.
- MTGNN [31]: This model leverages three temporal convolution modules with the dilation factor 1, and three graph convolution modules to capture spatio-temporal correlations. The depth and the retain ratio of a mix-hop propagation layer (in graph convolution module) are set to 2 and 0.05. The saturation rate of the activation function from the graph learning layer is set to 3. The size of node embeddings is 40. Besides, dropout with 0.3 is applied after each temporal convolution layer. Layernorm is applied after each graph convolution layer. The learning rate, l2 penalty, and the threshold of gradient clipping are set to $1e^{-3}$, $1e^{-4}$, and 5. Curriculum learning strategy is turned off.
- DCRNN [18]: This method applies an encoder-decoder architecture. Both encoder and decoder contain two recurrent layers and the number of hidden units is 64. The maximum steps of randoms walks is set to 2. The initial learning rate is $1e^{-2}$ and decreases to $\frac{1}{10}$ every 20 epochs starting from the 10th epochs. The threshold of gradient clipping is 5. For scheduled sampling, the probability decay is $\epsilon_i = \frac{\mu}{\mu + \exp(i/\mu)}$, where i is the number of training iterations and μ is the parameter to control the speed of convergence. μ is set to 2,000 on all datasets.
- AGCRN [1]: This approach uses a two-layer RNN to encoder the information and one linear layer to predict the future. The hidden units for all the AGCRN cells are set to 64. The embedding dimension is set as 10 for BAY and PEMS-04, and 2 for PEMS-08. The learning rate is $3e^{-3}$. Non-mentioned optimization tricks such as gradient clipping, are not applied.

Note that our reproduced results for the base models are close to, and some are even better than, the results reported in the original papers.

A.3 Settings of STGCL

For the settings described in Section A.2, STGCL uses the same, except that we don't apply weight decay, as this will introduce another term in the loss function.

Table 5: Complete results of the four base models and base models w/ JL-graph on the test set of PEMS-04, PEMS-08, and BAY datasets. The first, second, and third rows of each setting indicate MAE, RMSE, and MAPE, respectively.

Methods	PEMS-04			PEMS-08			BAY		
	15 min	30 min	60 min	15 min	30 min	60 min	15 min	30 min	60 min
GWN	18.20±.09	19.32±.13	21.10±.18	13.80±.05	14.75±.04	16.39±.09	1.31±.00	1.64±.01	1.97±.03
	29.15±.12	30.84±.15	33.38±.19	21.87±.04	23.71±.07	26.20±.11	2.75±.01	3.73±.04	4.55±.07
	12.67±.44	13.46±.38	15.05±.31	9.30±.42	9.88±.43	11.24±.51	2.71±.02	3.67±.03	4.62±.08
w/ JL-graph	17.93±.04	18.87±.04	20.40±.09	13.67±.04	14.61±.03	16.09±.05	1.29±.00	1.61±.00	1.88±.01
	28.86±.07	30.33±.07	32.47±.08	21.79±.04	23.66±.07	26.02±.12	2.72±.01	3.64±.01	4.34±.01
	12.36±.11	13.18±.25	14.61±.56	8.86±.16	9.74±.49	11.01±.58	2.70±.01	3.63±.02	4.46±.03
MTGNN	18.32±.05	19.10±.05	20.39±.09	14.36±.06	15.34±.10	16.91±.16	1.34±.02	1.66±.01	1.94±.02
	29.62±.13	31.15±.14	33.18±.16	22.47±.07	24.34±.08	26.75±.14	2.81±.01	3.76±.02	4.48±.03
	12.65±.20	13.14±.21	14.10±.21	9.38±.44	10.03±.25	11.50±.65	2.84±.09	3.74±.07	4.59±.08
w/ JL-graph	17.99±.03	18.72±.05	19.88±.07	14.04±.05	14.90±.05	16.23±.08	1.32±.00	1.63±.01	1.89±.01
	29.03±.08	30.34±.13	32.08±.13	22.16±.08	23.99±.10	26.23±.16	2.83±.01	3.76±.02	4.38±.02
	12.35±.10	12.92±.25	13.93±.52	9.16±.18	9.70±.17	10.49±.09	2.79±.02	3.68±.04	4.42±.04
DCRNN	19.99±.11	22.40±.19	27.15±.35	15.23±.15	16.98±.25	20.27±.41	1.34±.03	1.71±.05	2.09±.08
	31.49±.16	34.97±.29	41.64±.49	23.74±.21	26.65±.34	31.45±.55	2.83±.07	3.89±.08	4.85±.16
	13.48±.05	15.17±.12	18.74±.21	9.63±.07	10.73±.10	12.90±.21	2.81±.06	3.90±.10	5.05±.23
w/ JL-graph	19.82±.08	22.07±.12	26.51±.21	15.19±.11	16.89±.19	20.09±.34	1.32±.00	1.68±.00	2.05±.00
	31.26±.10	34.53±.17	40.85±.30	23.71±.14	26.61±.24	31.30±.40	2.79±.00	3.83±.01	4.76±.01
	13.44±.04	15.05±.06	18.44±.13	9.59±.03	10.70±.08	12.82±.19	2.77±.01	3.82±.04	4.92±.03
AGCRN	18.53±.03	19.43±.06	20.72±.03	14.58±.07	15.71±.07	17.82±.11	1.37±.00	1.69±.01	1.99±.01
	29.70±.13	31.29±.16	33.26±.13	22.75±.09	24.70±.13	27.76±.21	2.88±.01	3.85±.01	4.59±.02
	12.87±.44	13.23±.29	13.97±.14	9.61±.17	10.64±.49	12.17±.55	2.94±.02	3.85±.02	4.67±.02
w/ JL-graph	18.31±.04	19.17±.06	20.39±.03	14.51±.05	15.56±.06	17.51±.10	1.35±.00	1.67±.01	1.96±.01
	29.69±.20	31.23±.23	33.05±.18	22.69±.07	24.57±.12	27.47±.13	2.85±.01	3.80±.02	4.56±.02
	12.63±.18	13.16±.25	13.96±.13	9.61±.29	10.24±.28	11.47±.17	2.90±.02	3.79±.04	4.62±.08

For STGCL's specific settings, the temperature τ is set to 0.1 after tuning within the range of {0.05, 0.1, 0.15, 0.2, 0.25}. The augmentation method is fixed to input masking with a 1% ratio, since we empirically find that it generally performs best as shown in Section 4.4.1. The other two important hyper-parameters are λ and r_f. The applied values of them are in Table 6. We can observe that tuning λ within {0.01, 0.05, 0.1, 0.5, 1.0} is sufficient to find the best value, and r_f can set to 60 minutes as the default value.

Table 6: Values of the two important hyper-parameters (loss term trade-off parameter λ and negative filtering threshold r_f) used in each STGCL's reported result in Table 5.

Methods	PEMS-04		PEMS-08		BAY	
	λ	r_f	λ	r_f	λ	r_f
GWN w/ JL-node	0.05	60 min	0.05	60 min	0.05	60 min
GWN w/ JL-graph	0.5	60 min	0.05	60 min	1.0	60 min
MTGNN w/ JL-node	0.01	60 min	0.01	60 min	0.5	60 min
MTGNN w/ JL-graph	0.5	60 min	1.0	60 min	0.1	60 min
DCRNN w/ JL-node	0.5	60 min	0.1	60 min	0.05	60 min
DCRNN w/ JL-graph	0.1	60 min	0.05	60 min	0.01	60 min
AGCRN w/ JL-node	0.1	60 min	0.05	60 min	0.1	60 min
AGCRN w/ JL-graph	0.5	30 min	0.1	60 min	0.01	60 min

A.4 Settings of Two-stage Training

We use two models (GWN and AGCRN) to perform the pretraining & fine-tuning experiments. The model-related configurations are used as the same in Section A.2.

According to the typical pretraining procedure [34], we first train the STG encoder of the applied model with a contrastive objective. Based on the experience in Section A.3, we use input masking with a 1% ratio as the augmentation method and set the temperature τ and filtering threshold r_f to 0.1 and 60 minutes respectively. We optimize the encoder by Adam optimizer for a maximum of 100 epochs and use an early stop strategy with the patience of 10 by monitoring the training loss.

After training, we select the checkpoint with the lowest training loss to perform the downstream task. Specifically, we fine-tune the pretrained encoder with an untrained decoder to predict the future. We optimize the encoder and decoder by Adam optimizer, setting the learning rate to $1e^{-4}$ and $1e^{-3}$, respectively.