SWINBURNE UNIVERSITY OF TECHNOLOGY

Traffic Flow Prediction System

COS30018 - Intelligent System – Semester 2, 2024

NAME & STUDENT ID	**Nguyen Gia Binh – 104219428** **Pham Mai Hanh – 104198828** **Bui Tran Gia Bao – 104177225**
LECTURER & TUTOR	**PHAM THI KIM DUNG** **(**dtpham@swin.edu.vn)

Table of Contents

A. INTRODUCTION

Traffic congestion is a significant challenge in urban areas. It leads to increased travel times, fuel consumption, and pollution. Forecasting traffic flows is a critical issue for researchers and practitioners in the field of transportation. Accurate traffic flow prediction is essential for efficient traffic management and route optimization. However, this is particularly challenging since traffic data exhibits complex patterns due to various influencing factors such as the time of day, weather conditions, and road incidents. Traditional methods struggle to capture these complexities, necessitating advanced machine learning models.

1. Traffic Network

A traffic network can be visualized as an undirected graph denoted by G= (N, E, A), where N is the set of nodes, E denotes the edges, and A is the adjacency matrix. Each node in the network collects several measurements at a consistent frequency, generating a feature vector of length F at each time slice. These measurements include traffic flow, speed, and occupancy.

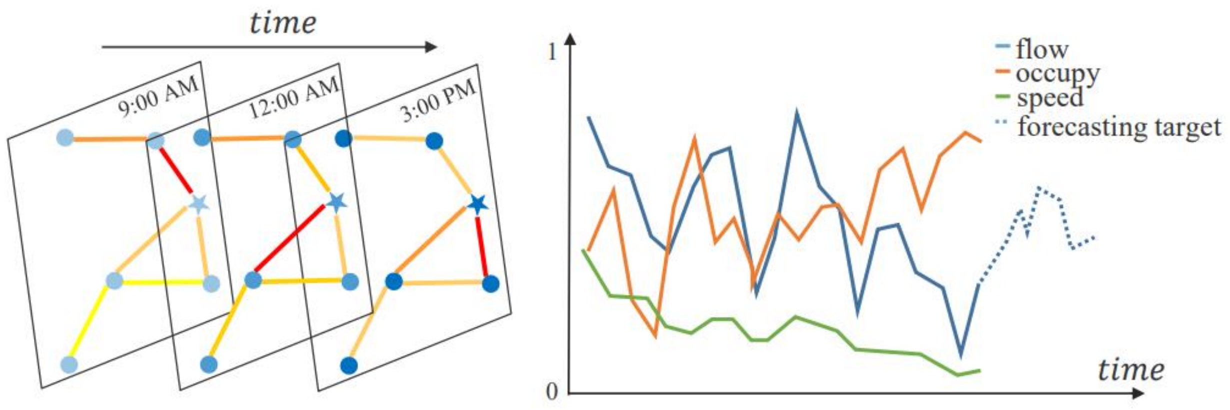


Figure 1: (left) The spatial-temporal structure of traffic data, where the data at each time slice forms a graph; (right) Three measurements are detected on a node and the future traffic flow is the forecasting target. All measurements are normalized to [0,1]

Looking at the left diagram, we see how the spatial influence of traffic flows varies at various times of the day. The bold line between the two points stands for their mutual influence strength. The darker the color of the line is, the greater the influence is. In the spatial dimension, we can find that various locations have different impacts on A and even the same location has varying influence on an as time goes by. At 9:00 AM, certain paths are heavily used due to, for instance, rush hour, while by 12 AM, the influence shifts to other paths which could be due to people going out on things like lunch break or for other issues.

The same phenomenon happens across time slices. Overall, this shows changes in traffic patterns throughout the day.

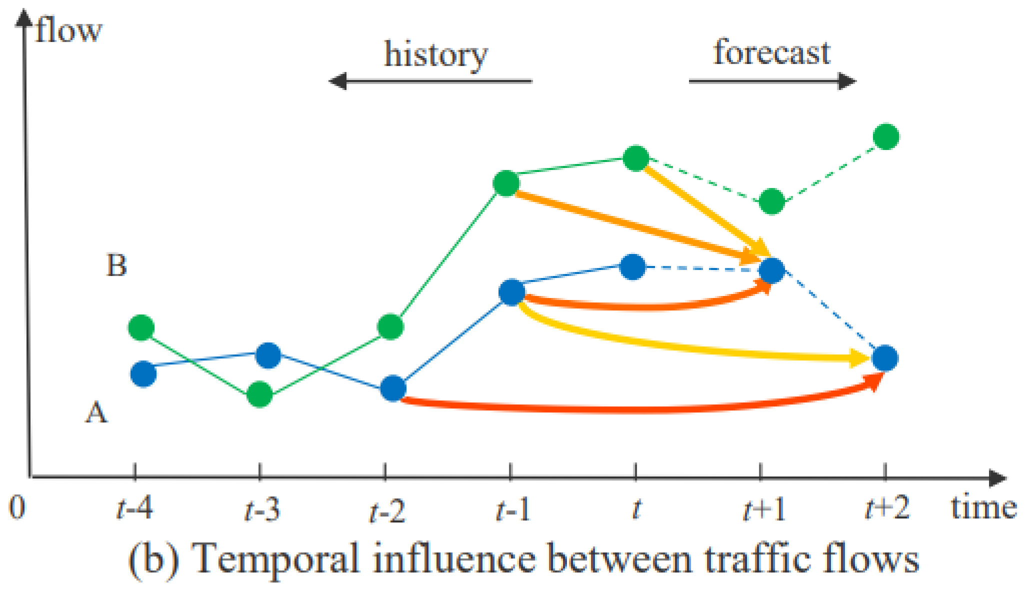


Figure 2: The temporal influence between traffic flows.

The temporal graph proves how the flow of traffic at one point in time is influenced by prior states. In the temporal dimension, the historical observations of various locations have varying impacts on traffic states at separate times in the future. For example, a traffic jam at 9:00 AM can influence the flow at subsequent times, potentially causing delays that ripple through the network.

The key points here are:

– The traffic data is not independent but highly correlated both spatially and temporally.

– The traffic at a particular node is influenced by traffic at adjacent nodes as well as the traffic conditions at previous time points.

⇒The challenge is to effectively extract these spatial-temporal correlations to make accurate traffic predictions.

2. Traffic Flow Forecasting Model (ASTGCN)

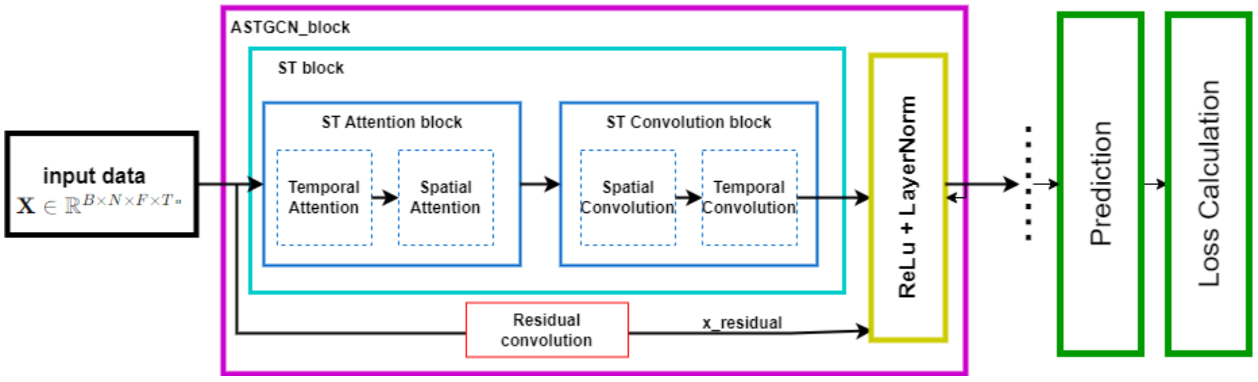


Figure 3: The graph of the overview of ASTGCN - Traffic Flow Forecasting Model

To tackle the complexity of traffic data and provide accurate traffic flow predictions, we chose an attention-based spatial-temporal graph convolutional network (ASTGCN) model. This model contains 2 major parts:

- **The spatial-temporal attention mechanism:** capture the dynamic spatial-temporal correlations in traffic data

- **The spatial-temporal convolution** simultaneously employs graph convolutions to capture the spatial patterns and common standard convolutions to describe the temporal features

The outputs are convolved and sliced to generate the final prediction results.

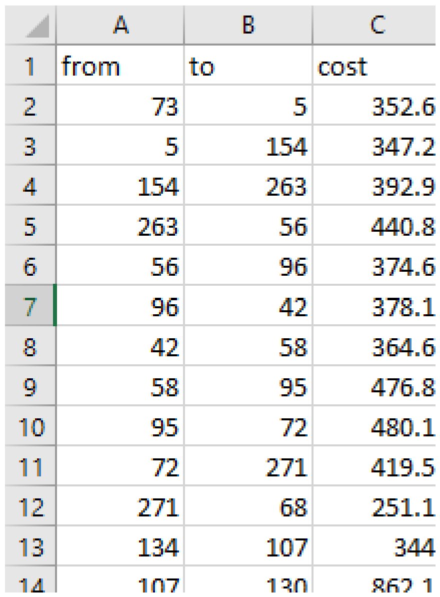
3. Dataset

The 2 real-world datasets from the Caltrans Performance Measurement System (PeMS) were used for predicting traffic with this model. The data was collected in real-time every 30 seconds and aggregated every 5 minutes, so each detector contains 288 data points per day. There are three kinds of traffic features to be considered including traffic flow, average speed, and average occupancy.

– Traffic flow refers to the number of vehicles passing a sensor (node) in the traffic network within a given time interval.
– Average speed indicates the average speed of vehicles passing the sensor during a given time interval.
– Average occupancy measures the percentage of time a sensor is occupied by a vehicle during a given time interval.

We used the dataset which was collected the most recently (in 2018) - PeMS04. This dataset contains traffic data collected in the San Francisco Bay Area, using 3848 detectors on 29 roads.

4. Raw Data Files

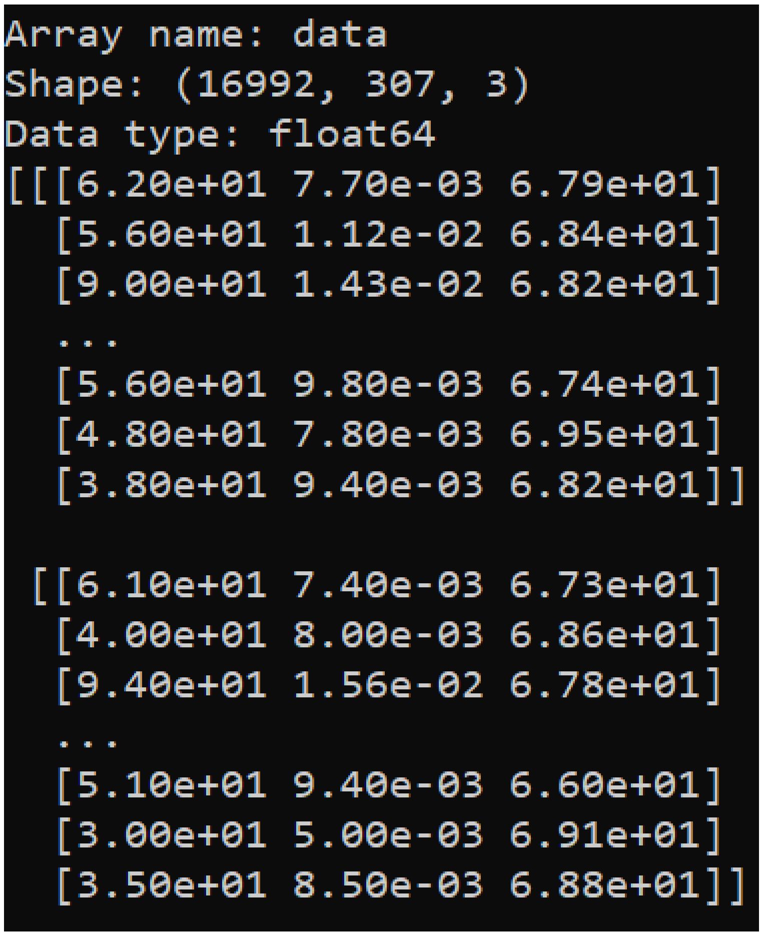


	A	B	C
1	from	to	cost
2	73	5	352.6
3	5	154	347.2
4	154	263	392.9
5	263	56	440.8
6	56	96	374.6
7	96	42	378.1
8	42	58	364.6
9	58	95	476.8
10	95	72	480.1
11	72	271	419.5
12	271	68	251.1
13	134	107	344
14	107	130	862.1

Figure 4: The Distance File in excel format.

Distance file contains the nodes' IDs and distances between traffic sensors (nodes) in the traffic network.

- **from:** The ID of the starting sensor.
- **to:** The ID of the ending sensor.
- **cost:** The physical distance/cost between the two sensors.



```
Array name: data
Shape: (16992, 307, 3)
Data type: float64
[[[6.20e+01 7.70e-03 6.79e+01]
  [5.60e+01 1.12e-02 6.84e+01]
  [9.00e+01 1.43e-02 6.82e+01]
  ...
  [5.60e+01 9.80e-03 6.74e+01]
  [4.80e+01 7.80e-03 6.95e+01]
  [3.80e+01 9.40e-03 6.82e+01]]

 [[6.10e+01 7.40e-03 6.73e+01]
  [4.00e+01 8.00e-03 6.86e+01]
  [9.40e+01 1.56e-02 6.78e+01]
  ...
  [5.10e+01 9.40e-03 6.60e+01]
  [3.00e+01 5.00e-03 6.91e+01]
  [3.50e+01 8.50e-03 6.88e+01]]
```

Figure 5: The PeMS04.npz file.

PeMS04.npz file contains the traffic data with dimensions (time steps, sensors, features) for each sensor over time.

- Shape: (16992, 307, 3)
    + 16992: Number of time steps.
    + 307: Number of nodes (sensors)
    + 3: Number of traffic features for each node
- Each entry in the array represents the traffic data at a particular time point for all sensors.
- The three values for each sensor are: traffic flow, average speed, and occupancy.

5. Data processing

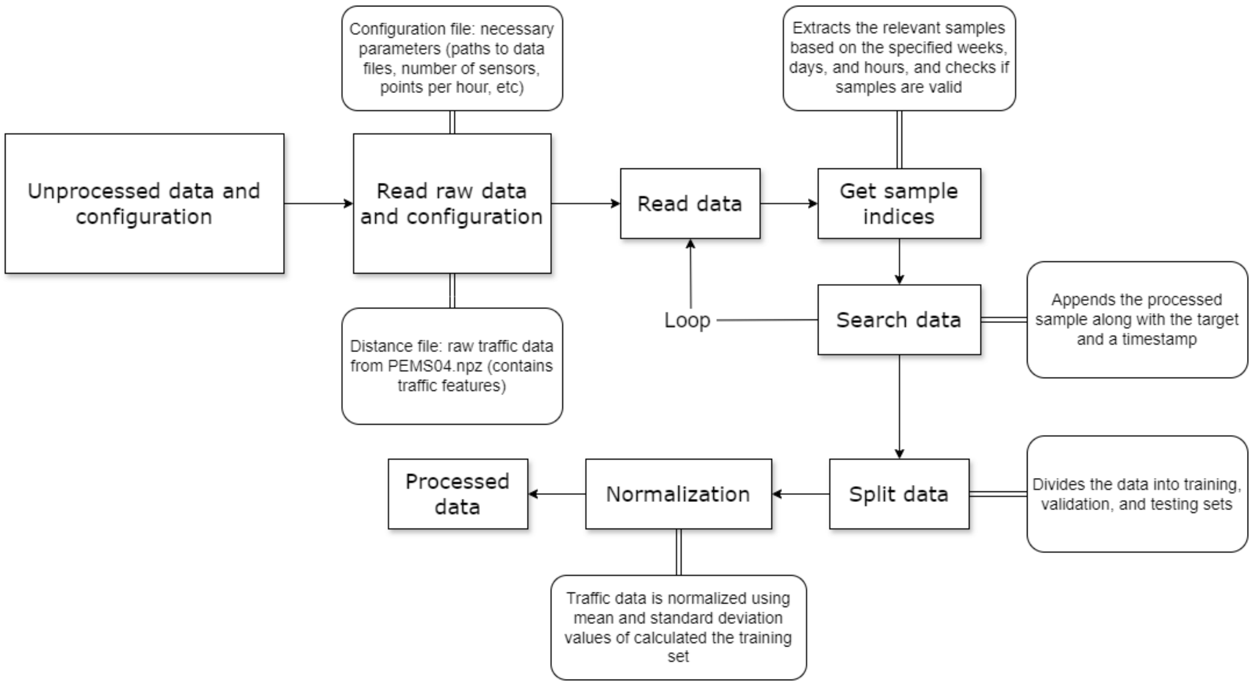

Configuration file: necessary parameters (paths to data files, number of sensors, points per hour, etc)
Extracts the relevant samples based on the specified weeks, days, and hours, and checks if samples are valid
Unprocessed data and configuration
Read raw data and configuration
Read data
Get sample indices
Loop
Search data
Appends the processed sample along with the target and a timestamp
Distance file: raw traffic data from PEMS04.npz (contains traffic features)
Processed data
Normalization
Split data
Divides the data into training, validation, and testing sets
Traffic data is normalized using mean and standard deviation values of calculated the training set

Figure 6: The graph about the flow of data processing.

a) Reading raw data and configuration

- **Configuration file (PEMS04_astgcn.conf):** The configuration file is read to obtain the necessary parameters for data processing. The most noticeable parameters in this file include:
  + num_of_vertices: Number of traffic sensors (nodes) in the dataset (307).
  + num_of_hours: Number of hours to consider (1).
  + points_per_hour: Number of data points per hour (12, for 5-minute intervals).
- **Distance file and PEMS04.npz file** was read to obtain raw traffic data and traffic features for processing.

b) Looping through each index in the dataset

- **Read data.**

  For each index in the dataset, input samples are created based on historical data defined by weeks, days, and hours. The data for each sensor (node) is read and prepared for creating input-output pairs for training.

- **Get sample indices.**

  Next, it extracts relevant indices for weekly, daily, and hourly data based on the provided parameters, ensuring samples are valid.

- **Search data.**

  It then appends the processed sample along with the target (the value to be predicted) and a timestamp to the dataset.

The loop continues to process the next index in the dataset until all data points are processed.

c) Splitting data

After the loop, the traffic data gets divided into training, validation, and testing sets. This splitting is performed to ensure distinct data sets for separate phases of model evaluation.

d) Normalizing data

Traffic data is normalized using mean and standard deviation values of calculated the training set. (This involves adjusting the data by the mean and standard deviation of the training set to ensure that model inputs are scaled appropriately.)

Finally, it saves the processed data, including the normalized sequences, mean, and standard deviation into a new .npz file. The processed data file contains the normalized input sequences, target sequences, and timestamps, ready for training the ASTGCN model.

B. ATTENTION MECHANISM

Attention mechanisms have become crucial in deep learning models, particularly for tasks involving sequential or structured data (Vaswani et al., 2017). In spatial-temporal models like ASTGCN, attention helps the model focus on the most relevant information across both space and time, which is particularly important in traffic prediction where the influence of separate locations and time periods can vary significantly (Guo et al., 2019).

The attention mechanism allows the model to adaptively assign different weights to various inputs, effectively "paying attention" to the most informative parts of the data. This dynamic weighting is key to capturing the complex, non-stationary nature of traffic patterns (Li et al., 2018).

1. Spatial Attention

Spatial attention in ASTGCN allows the model to focus on the most influential nodes for each prediction, adapting to changing traffic patterns. The mathematical formulation of spatial attention is:

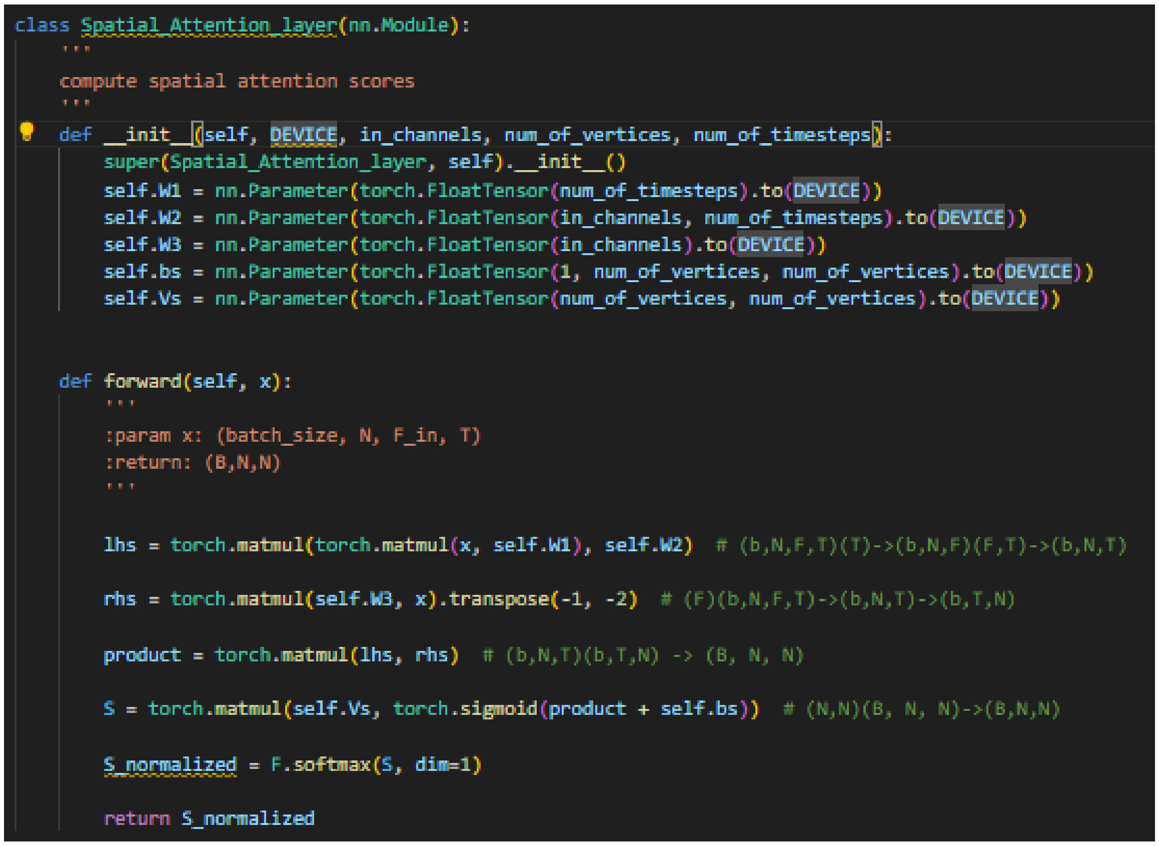
$$\mathbf{S} = \mathbf{V}_s \cdot \sigma((\boldsymbol{\mathcal{X}}_h^{(r-1)}\mathbf{W}_1)\mathbf{W}_2(\mathbf{W}_3\boldsymbol{\mathcal{X}}_h^{(r-1)})^T + \mathbf{b}_s)$$

Figure 7: The Spatial Attention formulation.

Where S is the spatial attention score matrix, Vs, W1, W2, W3 are learnable parameter matrices, bs is a learnable bias term, σ is a sigmoid activation function, and X is the input feature matrix.

This formulation allows the model to compute a unique attention score for each pair of nodes in the graph, effectively learning the relative importance of different spatial relationships (Zhang et al., 2018).

Spatial attention is implemented in code. Here it is:



```python
class Spatial_Attention_layer(nn.Module):
    '''
    compute spatial attention scores
    '''
    def __init__(self, DEVICE, in_channels, num_of_vertices, num_of_timesteps):
        super(Spatial_Attention_layer, self).__init__()
        self.W1 = nn.Parameter(torch.FloatTensor(num_of_timesteps).to(DEVICE))
        self.W2 = nn.Parameter(torch.FloatTensor(in_channels, num_of_timesteps).to(DEVICE))
        self.W3 = nn.Parameter(torch.FloatTensor(in_channels).to(DEVICE))
        self.bs = nn.Parameter(torch.FloatTensor(1, num_of_vertices, num_of_vertices).to(DEVICE))
        self.Vs = nn.Parameter(torch.FloatTensor(num_of_vertices, num_of_vertices).to(DEVICE))


    def forward(self, x):
        '''
        :param x: (batch_size, N, F_in, T)
        :return: (B,N,N)
        '''

        lhs = torch.matmul(torch.matmul(x, self.W1), self.W2)  # (b,N,F,T)(T)->(b,N,F)(F,T)->(b,N,T)

        rhs = torch.matmul(self.W3, x).transpose(-1, -2)  # (F)(b,N,F,T)->(b,N,T)->(b,T,N)

        product = torch.matmul(lhs, rhs)  # (b,N,T)(b,T,N) -> (B, N, N)

        S = torch.matmul(self.Vs, torch.sigmoid(product + self.bs))  # (N,N)(B, N, N)->(B,N,N)

        S_normalized = F.softmax(S, dim=1)

        return S_normalized
```

Figure 8: The S-A implemented in code.

2. Temporal Attention

Temporal attention helps the model identify and prioritize the most relevant historical time steps for making predictions. The mathematical formulation of temporal attention is:

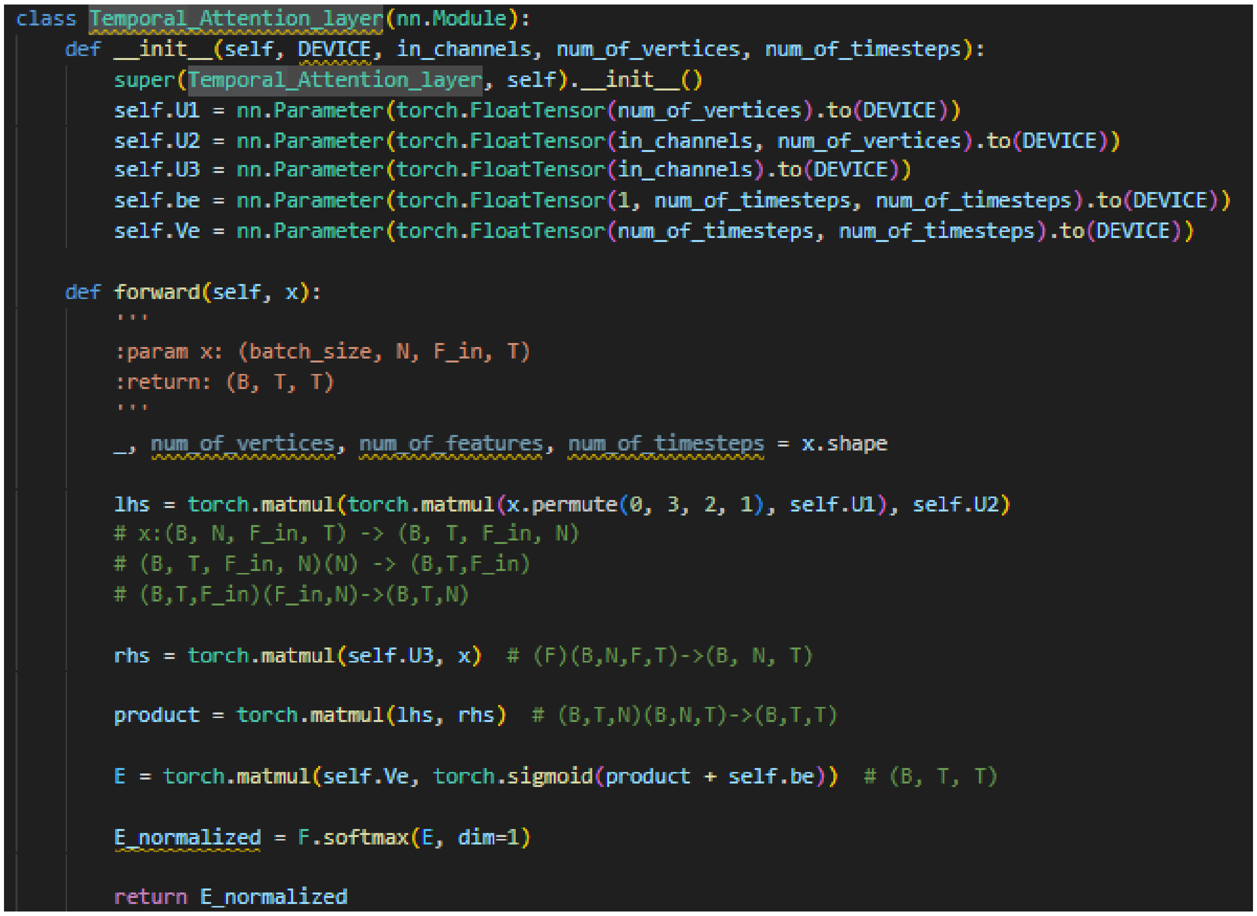
$$\mathbf{E} = \mathbf{V}_e \cdot \sigma(((\boldsymbol{\mathcal{X}}_h^{(r-1)})^T \mathbf{U}_1)\mathbf{U}_2(\mathbf{U}_3 \boldsymbol{\mathcal{X}}_h^{(r-1)}) + \mathbf{b}_e)$$

Figure 9: The Temporal Attention formulation.

Where E is the temporal attention score matrix, Ve, U1, U2, U3 are learnable parameter matrices, be is a learnable bias term, σ is a sigmoid activation function, and X is the input feature matrix.

This formulation allows the model to compute attention scores between different time steps, learning to focus on the most predictive historical patterns (Liang et al., 2018).

The implementation of temporal attention in the code:



```
class Temporal_Attention_layer(nn.Module):
    def __init__(self, DEVICE, in_channels, num_of_vertices, num_of_timesteps):
        super(Temporal_Attention_layer, self).__init__()
        self.U1 = nn.Parameter(torch.FloatTensor(num_of_vertices).to(DEVICE))
        self.U2 = nn.Parameter(torch.FloatTensor(in_channels, num_of_vertices).to(DEVICE))
        self.U3 = nn.Parameter(torch.FloatTensor(in_channels).to(DEVICE))
        self.be = nn.Parameter(torch.FloatTensor(1, num_of_timesteps, num_of_timesteps).to(DEVICE))
        self.Ve = nn.Parameter(torch.FloatTensor(num_of_timesteps, num_of_timesteps).to(DEVICE))

    def forward(self, x):
        '''
        :param x: (batch_size, N, F_in, T)
        :return: (B, T, T)
        '''
        _, num_of_vertices, num_of_features, num_of_timesteps = x.shape

        lhs = torch.matmul(torch.matmul(x.permute(0, 3, 2, 1), self.U1), self.U2)
        # x:(B, N, F_in, T) -> (B, T, F_in, N)
        # (B, T, F_in, N)(N) -> (B,T,F_in)
        # (B,T,F_in)(F_in,N)->(B,T,N)

        rhs = torch.matmul(self.U3, x)  # (F)(B,N,F,T)->(B, N, T)

        product = torch.matmul(lhs, rhs)  # (B,T,N)(B,N,T)->(B,T,T)

        E = torch.matmul(self.Ve, torch.sigmoid(product + self.be))  # (B, T, T)

        E_normalized = F.softmax(E, dim=1)

        return E_normalized
```

Figure 10: The T-S implemented in code.

3. The graph visualization of spatial-temporal attention

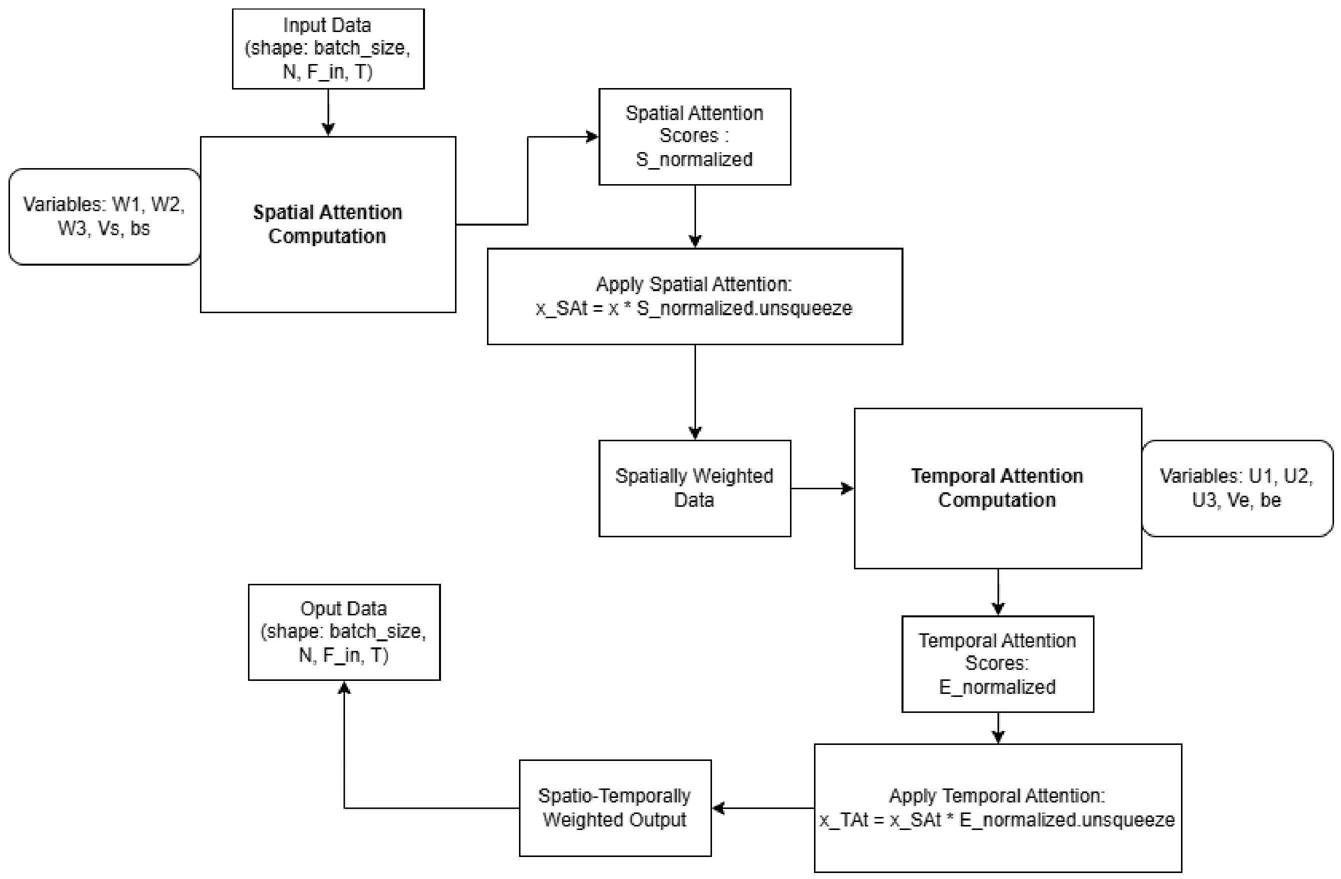

Input Data (shape: batch_size, N, F_in, T)
Variables: W1, W2, W3, Vs, bs
Spatial Attention Computation
Spatial Attention Scores : S_normalized
Apply Spatial Attention: x_SAt = x * S_normalized.unsqueeze
Spatially Weighted Data
Temporal Attention Computation
Variables: U1, U2, U3, Ve, be
Temporal Attention Scores: E_normalized
Apply Temporal Attention: x_TAt = x_SAt * E_normalized.unsqueeze
Spatio-Temporally Weighted Output
Oput Data (shape: batch_size, N, F_in, T)

Figure 11: The graph represents S-T Attention

The ASTGCN model employs a dual-attention mechanism to process traffic data, enhancing prediction accuracy by focusing on both spatial and temporal dimensions.

The process begins with spatial attention, where the model computes and normalizes attention scores to weight input features based on their spatial relevance. This is followed by temporal attention, which further refines the spatially weighted data by emphasizing temporally significant information.

The result is a spatial-temporally weighted output that maintains the original input's shape but captures the most informative aspects of both spatial locations and time periods in the traffic network. This adaptive approach allows ASTGCN to dynamically adjust its focus, potentially leading to more nuanced and accurate traffic predictions by efficiently processing complex, multidimensional traffic data.

C. SPATIAL-TEMPORAL MECHANISM

Traditional convolutional neural networks (CNNs) are designed for grid-structured data and cannot be directly applied to graph-structured data like traffic networks (Bruna et al., 2014). Graph convolution addresses this limitation by generalizing the convolution operation to graph-structured data.

The concept of spectral graph convolution is based on graph signal processing theory, defining the convolution operation in the spectral domain of the graph Laplacian (Hammond et al., 2011).

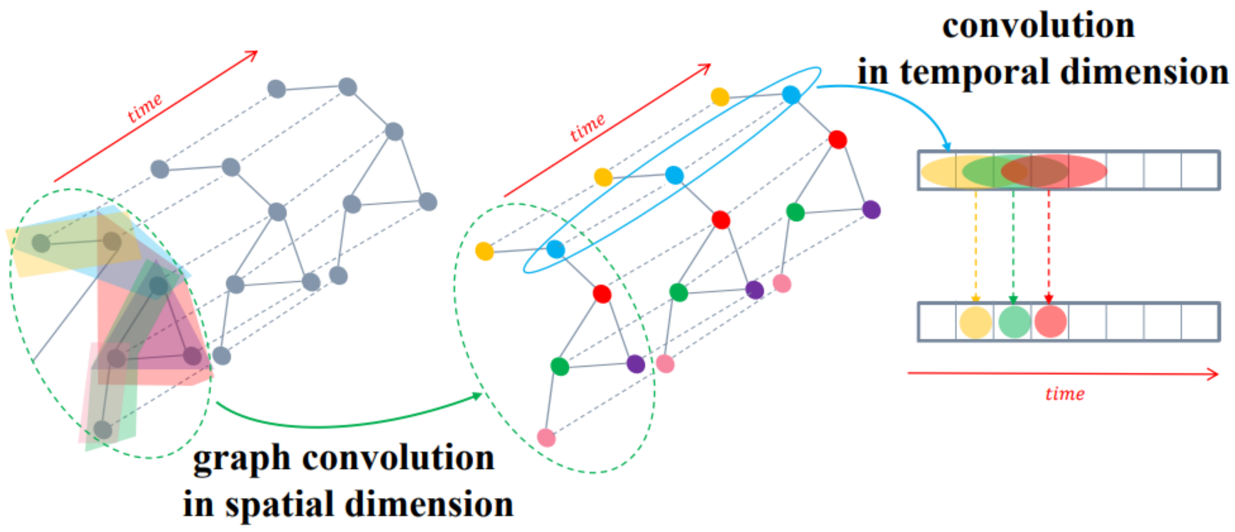


Figure 12: Both graph convolution in spatial dimension and temporal dimension

1. Chebyshev Graph Convolution

ASTGCN uses the Chebyshev polynomial approximation to make spectral graph convolution computationally efficient. The mathematical formulation is:

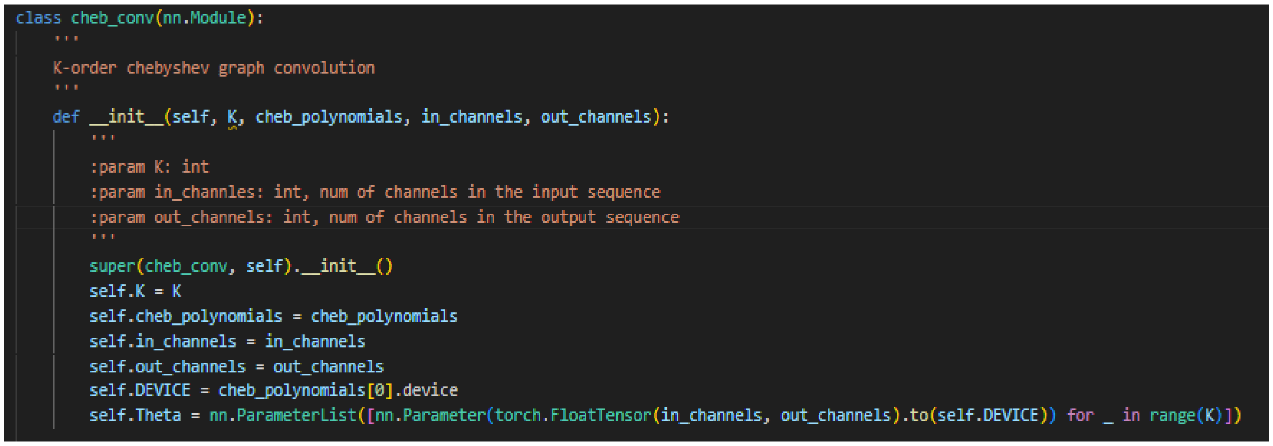
$$g_\theta *_G x = g_\theta(\mathbf{L})x = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\mathbf{L}})x$$

Figure 13: The Chebyshev polynomial approximation formulation.

Where gθ is the filter, x is the input signal, θk are learnable parameters, Tk are Chebyshev polynomials, and L̃ is the scaled and normalized graph Laplacian.

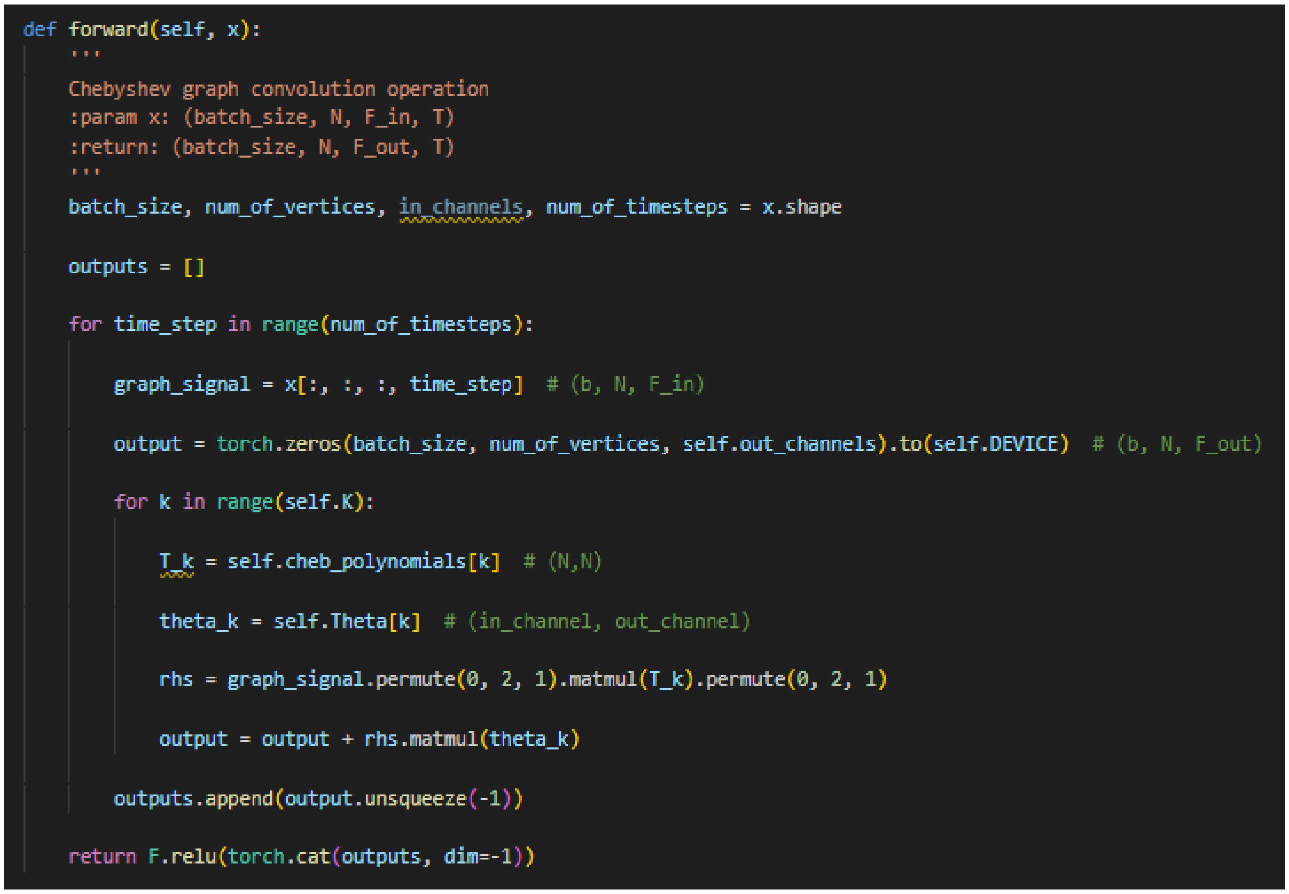
This approximation allows for localized filters and reduces the computational complexity from O(n^2) to O(K|E|), where n is the number of nodes, K is the polynomial order, and |E| is the number of edges (Defferrard et al., 2016).

The implementation of Chebyshev graph convolution in the code:



```
class cheb_conv(nn.Module):
    '''
    K-order chebyshev graph convolution
    '''
    def __init__(self, K, cheb_polynomials, in_channels, out_channels):
        '''
        :param K: int
        :param in_channles: int, num of channels in the input sequence
        :param out_channels: int, num of channels in the output sequence
        '''
        super(cheb_conv, self).__init__()
        self.K = K
        self.cheb_polynomials = cheb_polynomials
        self.in_channels = in_channels
        self.out_channels = out_channels
        self.DEVICE = cheb_polynomials[0].device
        self.Theta = nn.ParameterList([nn.Parameter(torch.FloatTensor(in_channels, out_channels).to(self.DEVICE)) for _ in range(K)])
```

Figure 14: The def __init__ of Chebyshev Graph Convolution



```python
def forward(self, x):
    '''
    Chebyshev graph convolution operation
    :param x: (batch_size, N, F_in, T)
    :return: (batch_size, N, F_out, T)
    '''
    batch_size, num_of_vertices, in_channels, num_of_timesteps = x.shape

    outputs = []

    for time_step in range(num_of_timesteps):

        graph_signal = x[:, :, :, time_step]  # (b, N, F_in)

        output = torch.zeros(batch_size, num_of_vertices, self.out_channels).to(self.DEVICE)  # (b, N, F_out)

        for k in range(self.K):

            T_k = self.cheb_polynomials[k]  # (N,N)

            theta_k = self.Theta[k]  # (in_channel, out_channel)

            rhs = graph_signal.permute(0, 2, 1).matmul(T_k).permute(0, 2, 1)

            output = output + rhs.matmul(theta_k)

        outputs.append(output.unsqueeze(-1))

    return F.relu(torch.cat(outputs, dim=-1))
```

Figure 15: The def forward of Chebyshev Graph Convolution

The Chebyshev graph convolution process in the ASTGCN model integrates spatial attention to enhance its ability to capture complex spatial-temporal dependencies in traffic networks.

The process begins with an input tensor representing the traffic data across multiple nodes, features, and time steps, along with pre-computed spatial attention scores. As the convolution unfolds, it iterates through each time step, extracting the graph signal and applying various orders of Chebyshev polynomials. These polynomials are then multiplied with the spatial attention scores, effectively weighing the importance of different spatial relationships. Learnable Theta parameters are applied to this weighted result, allowing the model to adapt to specific patterns in the data. The outcomes from all time steps are accumulated and combined, followed by the application of a ReLU activation function.

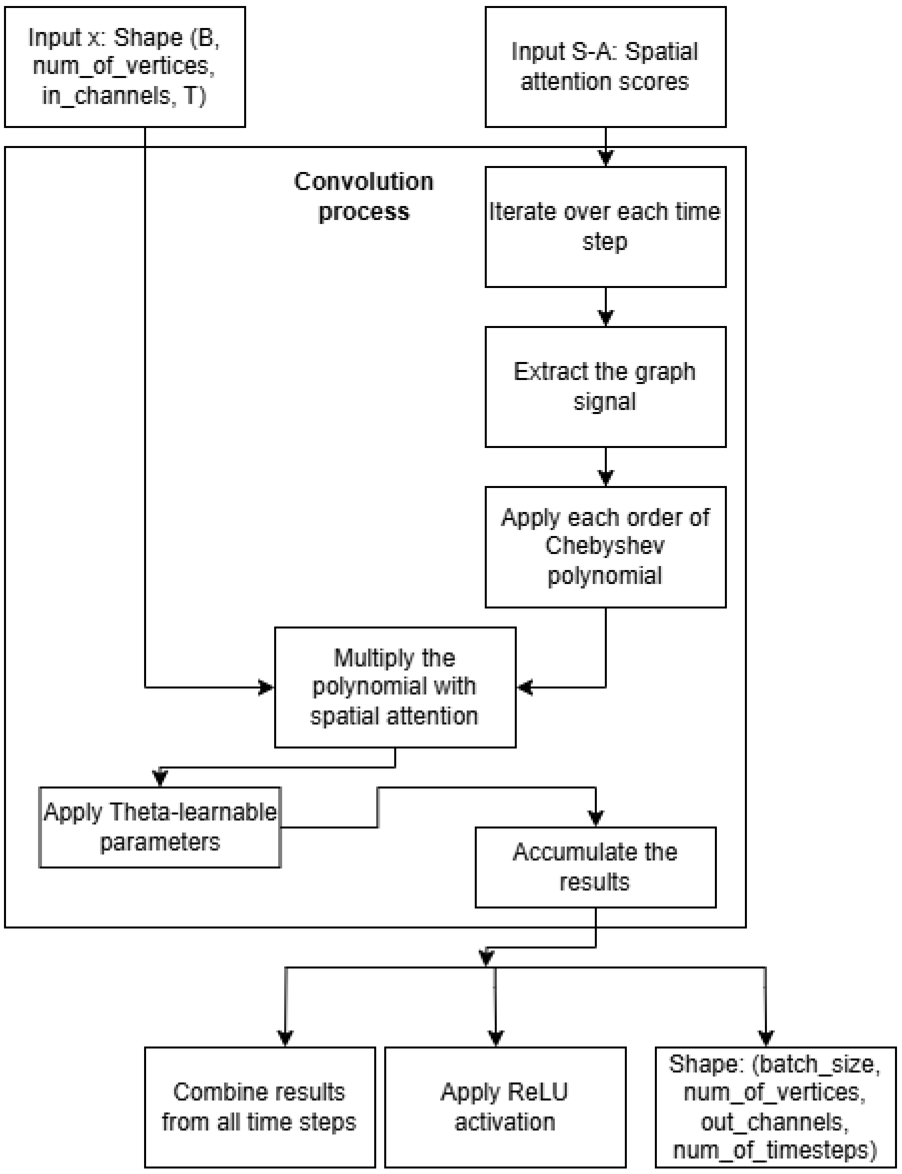
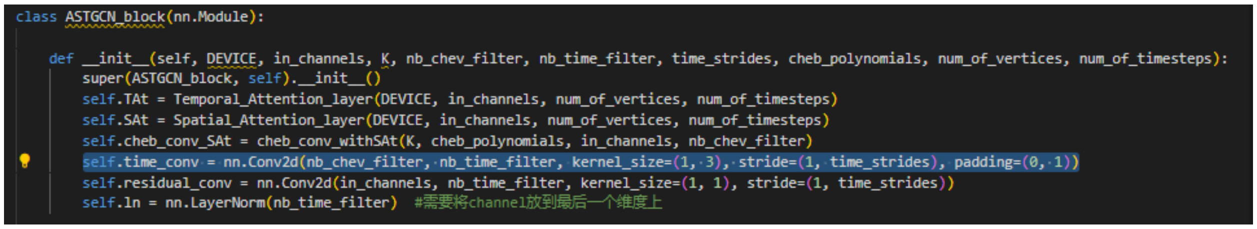

Input x: Shape (B, num_of_vertices, in_channels, T)
Input S-A: Spatial attention scores
Convolution process
Iterate over each time step
Extract the graph signal
Apply each order of Chebyshev polynomial
Multiply the polynomial with spatial attention
Apply Theta-learnable parameters
Accumulate the results
Combine results from all time steps
Apply ReLU activation
Shape: (batch_size, num_of_vertices, out_channels, num_of_timesteps)

Figure 16: The graph of Chebyshev Graph Convolution

This sophisticated process yields an output that maintains the temporal dimension while transforming the input features, enabling the model to effectively capture and leverage the most relevant spatial-temporal patterns for accurate traffic prediction.

2. Temporal Convolution

Temporal convolution in ASTGCN captures dependencies along the time axis. It's implemented as a standard 1D convolution operation over the time dimension, using a kernel size of 3 to capture short-term temporal patterns.



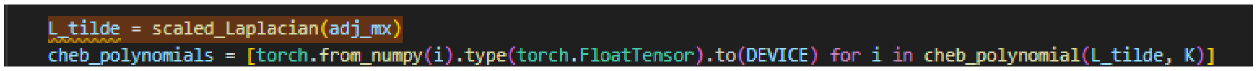
```python
class ASTGCN_block(nn.Module):

    def __init__(self, DEVICE, in_channels, K, nb_chev_filter, nb_time_filter, time_strides, cheb_polynomials, num_of_vertices, num_of_timesteps):
        super(ASTGCN_block, self).__init__()
        self.TAt = Temporal_Attention_layer(DEVICE, in_channels, num_of_vertices, num_of_timesteps)
        self.SAt = Spatial_Attention_layer(DEVICE, in_channels, num_of_vertices, num_of_timesteps)
        self.cheb_conv_SAt = cheb_conv_withSAt(K, cheb_polynomials, in_channels, nb_chev_filter)
        self.time_conv = nn.Conv2d(nb_chev_filter, nb_time_filter, kernel_size=(1, 3), stride=(1, time_strides), padding=(0, 1))
        self.residual_conv = nn.Conv2d(in_channels, nb_time_filter, kernel_size=(1, 1), stride=(1, time_strides))
        self.ln = nn.LayerNorm(nb_time_filter)  #需要将channel放到最后一个维度上
```

Figure 17: The temporal Convolution implemented in code.

3. The Integration of Spatial and Temporal Convolutions

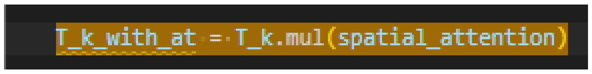
The integration of spatial and temporal convolutions in ASTGCN involves:

– Chebyshev polynomial computation: The Chebyshev polynomials are pre-computed and passed to the cheb_conv_withSAt class.



```python
L_tilde = scaled_Laplacian(adj_mx)
cheb_polynomials = [torch.from_numpy(i).type(torch.FloatTensor).to(DEVICE) for i in cheb_polynomial(L_tilde, K)]
```

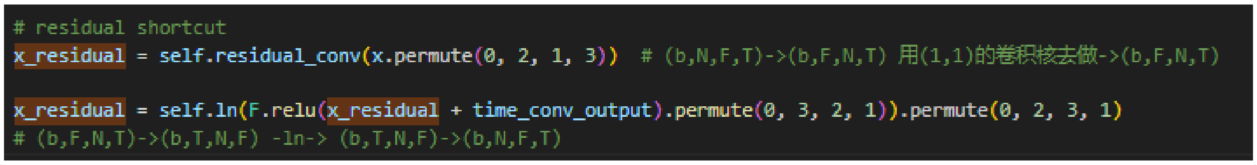
– Integration with attention: The spatial attention is directly incorporated into the Chebyshev graph convolution (cheb_conv_withSAt class)



```python
T_k_with_at = T_k.mul(spatial_attention)
```

– Multi-scale temporal convolution: The model uses multiple ASTGCN blocks with different temporal convolution strides to capture patterns at different time scales (ASTGCN_block class) in the code of Temporal Convolution part.

– Residual connections: Being used to facilitate gradient flow and enable training of deeper networks (ASTGCN_block class)



```python
# residual shortcut
x_residual = self.residual_conv(x.permute(0, 2, 1, 3))  # (b,N,F,T)->(b,F,N,T) 用(1,1)的卷积核去做->(b,F,N,T)

x_residual = self.ln(F.relu(x_residual + time_conv_output).permute(0, 3, 2, 1)).permute(0, 2, 3, 1)
# (b,F,N,T)->(b,T,N,F) -ln-> (b,T,N,F)->(b,N,F,T)
```

This integration allows the model to capture complex spatial-temporal patterns at different scales while maintaining computational efficiency.

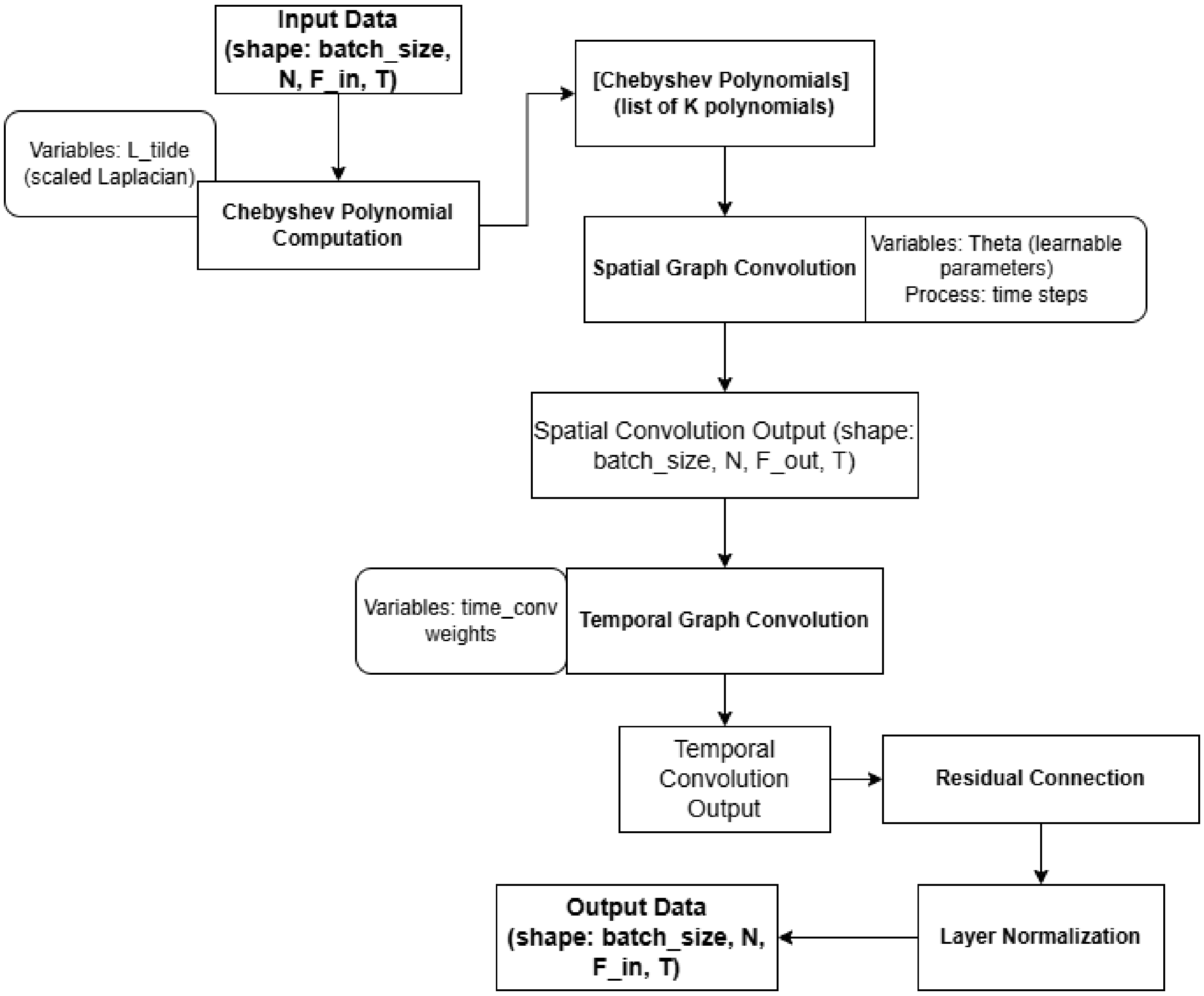

Input Data (shape: batch_size, N, F_in, T)
Variables: L_tilde (scaled Laplacian)
Chebyshev Polynomial Computation
[Chebyshev Polynomials] (list of K polynomials)
Spatial Graph Convolution
Variables: Theta (learnable parameters) Process: time steps
Spatial Convolution Output (shape: batch_size, N, F_out, T)
Variables: time_conv weights
Temporal Graph Convolution
Temporal Convolution Output
Residual Connection
Layer Normalization
Output Data (shape: batch_size, N, F_in, T)

Figure 18: The graph represents the flow of S-T Graph Convolution

The ASTGCN model employs a sophisticated spatial-temporal convolution process to analyze traffic data.

Initially, the input data, structured as a four-dimensional tensor, undergoes Chebyshev polynomial computation using a scaled Laplacian matrix. These polynomials are then utilized in a spatial graph convolution step, incorporating learnable Theta parameters to process data across multiple time steps. The output of this spatial convolution, maintaining the input's dimensionality but with transformed features, is subsequently fed into a temporal graph convolution. This step employs specific time_conv weights to capture temporal patterns. The resulting temporal convolution output is enhanced through a residual connection and refined by layer normalization. This intricate process preserves the input's original shape while effectively extracting and integrating both spatial and temporal dependencies from the traffic network.

By doing so, the ASTGCN model can learn and use complex spatial-temporal patterns, leading to more accurate and nuanced traffic predictions.

4. The Relationship Between Attention Mechanism and Spatial-Temporal Convolution

The relationship between the attention mechanism and the spatial-temporal convolution in the ASTGCN model can be summarized as:

- Sequential Processing: The attention mechanism is applied before the convolution.
- Dynamic Feature Weighting: Attention adjusts the adjacency matrix and input data.
- Complementary Roles: Attention focuses on relevant information, while convolution extracts features.
- Enhanced Feature Extraction: Attention guides convolution to focus on important relationships.
- Adaptive Learning: The combination allows the model to adapt to changing traffic patterns.

This constructive collaboration enables the model to capture complex, dynamic spatial-temporal dependencies in traffic data more effectively than using either component alone (Guo et al., 2019).

D. FINAL CONVOLUTION

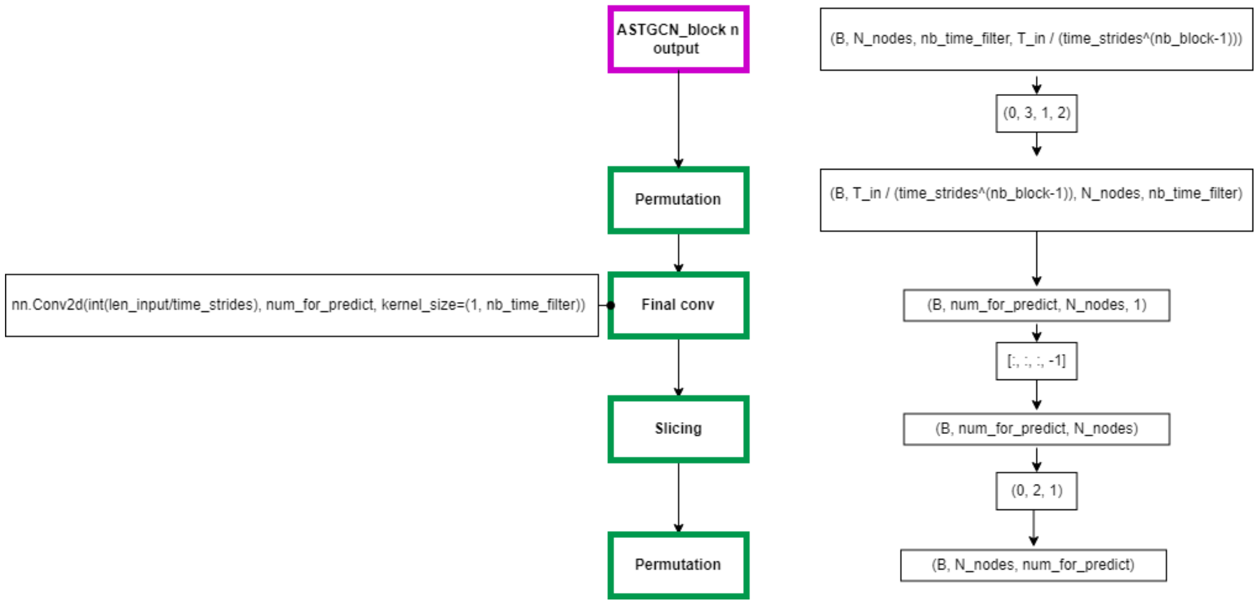

ASTGCN_block n output
(B, N_nodes, nb_time_filter, T_in / (time_strides^(nb_block-1)))
(0, 3, 1, 2)
Permutation
(B, T_in / (time_strides^(nb_block-1)), N_nodes, nb_time_filter)
nn.Conv2d(int(len_input/time_strides), num_for_predict, kernel_size=(1, nb_time_filter))
Final conv
(B, num_for_predict, N_nodes, 1)
[:, :, :, -1]
Slicing
(B, num_for_predict, N_nodes)
(0, 2, 1)
Permutation
(B, N_nodes, num_for_predict)

Figure 19: The graph represents the Final Convolution

The output from the last ASTGCN block is a tensor with dimensions:
(B, N_nodes, nb_time_filter, T_in / (time_strides^(nb_block 1))). This tensor represents the features extracted by the network after processing through several layers and blocks:

- B: the batch size
- N_nodes: the number of nodes in the graph (each node corresponding to a location or sensor in the traffic network),
- Nb_time_filter: the number of temporal filters applied,
- T_in / (time_strides^(nb_block-1)): the temporal dimension adjusted by the number of blocks and the stride in each.

Before applying the final convolution, the dimensions of the output tensor are permuted to align with the requirements of the convolution operation. The tensor is transformed from (B, N_nodes, nb_time_filter, T) to (B, T, N_nodes, nb_time_filter), where T denotes the adjusted time dimension after the blocks.

A 2D convolution is applied using nn.Conv2d with a kernel size of (1, nb_time_filter), aiming to map the multi-channel feature maps to a desired number of prediction outputs (num_for_predict). The convolution compresses the temporal and filter dimensions into the predicted output dimension, resulting in a tensor of shape (B, num_for_predict, N_nodes, 1). The last dimension is reduced to 1 because the kernel size of the convolution spans the entire nb_time_filter dimension. Each feature map from the convolution provides a prediction for a future time step at each node.

After the convolution, the last dimension (which is singleton due to the kernel size matching the number of input filters) is removed by slicing. This operation simplifies the tensor from four dimensions to three, resulting in a shape of (B, num_for_predict, N_nodes).

The tensor is permuted one final time to match the expected output format of the model. This permutation rearranges the tensor from (B, num_for_predict, N_nodes) to (B, N_nodes, num_for_predict).

E. OPTIMIZATION AND PREDICTION

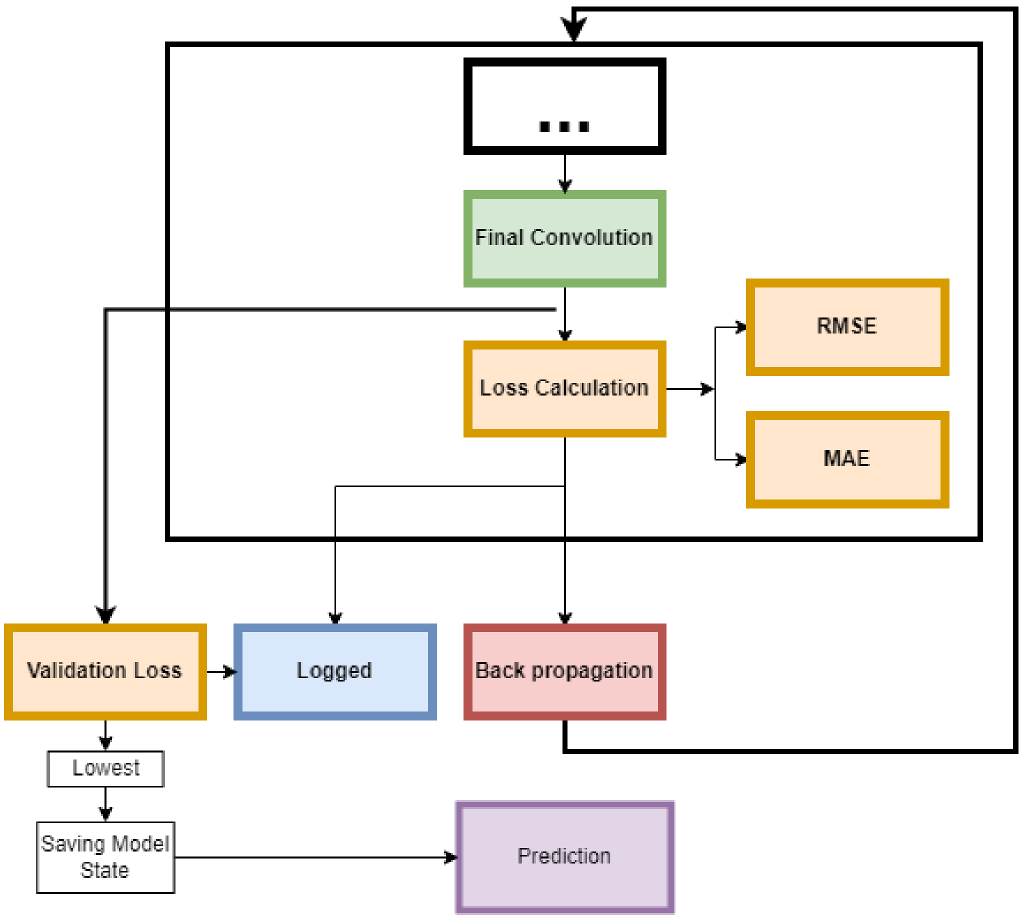


Figure 20: The graph represents the Optimization and Prediction

The outputs from the Final Convolution are compared to the true labels. This is where the actual numerical loss value is computed, indicating how well the model’s predictions match the true labels.

After the loss is calculated, backpropagation is triggered. This involves computing the gradients of the loss with respect to all trainable parameters in the model. The gradients are used by the optimizer to update the weights, aiming to minimize the loss in subsequent iterations. This process repeats for each batch, and loss is logged periodically using TensorBoard (SummaryWriter).

After each epoch, the model's performance is evaluated on a validation set. The validation process does not update model weights but provides a validation loss metric to gauge the model’s generalization ability. If the validation loss is the lowest observed, the model's state is saved for potential use on unseen test data.

Once the model is trained, it's set to evaluation mode (net.train(False)) to ensure that operations such as dropout are disabled. The best-performing model weights from the training phase are loaded to ensure predictions are made from the most accurate version of the model. The test dataset is passed through the model and for each batch, the model predicts outputs based on the learned weights, without any further adjustment to the model (torch.no_grad() is used to prevent gradient computation). Predictions and actual targets are collected across all test batches. Predictions are saved to a NumPy file for later analysis or deployment usage.

F. MODEL COMPARISION

1. Models Structure and Capabilities

FEATURE	ASTGCN (2019)	ASTGNN (2021)	DSTAGNN (2022)
Spatial-Temporal Attention	Only basic attention mechanism to model spatial and temporal correlations separately within traffic data.	Adaptively focus on varying spatial-temporal correlations, providing a more responsive model to sudden changes in traffic conditions.	Hierarchical Multi-Scale Attention that allows the model to adjust its focus at diverse levels of granularity depending on the context.
Handling of External Influences	Lacks integration of external factors such as weather, events, or road conditions that can significantly influence traffic patterns.	Incorporates external factors like weather conditions into the prediction model	By integrating a wide range of external data inputs, such as weather, events, and even socio-economic data, adapting predictions in real-time to these factors.
Graph Neural Network Techniques	Only basic graph convolutions with limited adaptability to network changes.	Utilizes more advanced graph neural network techniques that improve the model's ability to learn from complex spatial relationships and interactions within the traffic network.	Enhanced Deep Graph Convolutional Networks that include deeper layers and specialized node processing for complex traffic networks

Modeling of Traffic Patterns	Static modeling of traffic patterns based on historical data without adapting to anomalies or non-periodic events.	Consider both periodic trends and non-periodic anomalies, making the system more adaptable to unusual traffic conditions.	Continuous Time Series Analysis that integrates non-linear and non-stationary data processing for predicting highly volatile traffic patterns.
Learning and Adaptation	Operates with a fixed learning model that does not adjust post-deployment, reducing its effectiveness as traffic patterns evolve.	Learns continually from incoming data, enhancing its long-term accuracy and adaptability to new traffic trends.	Continuously refine and adjust its predictions based on live data, ensuring optimal performance even as traffic dynamics evolve drastically.

2. Models Configuration and Set-up

Due to multiple elements that are set up differently in the 2 models that we are going to compare ASTGCN against, minor re-configuration and set-up are required for the fairest comparison in our ability:

1. Open Anaconda Prompt (Windows) or terminal (Mac/Linux)
2. Create a new environment: conda create -n <name your created env> python=3.8
3. Activate the environment: conda activate <name your created env>
4. Install required packages (for each model required)
5. Follow the repository's README for running instructions.

Models:

- ASTGCN (2019): The latest version available on GitHub by the authors
- AGTGNN (2021): The latest version available on GitHub by the authors
- DSTAGNN (2022): base version since no newer version posted by authors.

Data splitting: Using the baseline 6:2:2 ratio on the PemS04 dataset.

Config file:

- Prediction window: num_for_predict = 12 (1 hour)
- Number of terms of the Chebyshev polynomial (equal to the number of spatial attention heads): K = 3
- epochs = 30
- learning_rate = 0.001

Training file: Adam optimizer

Devices to run: Lenovo Legion Slim 5 with:

- CPU: Intel Core i7-13700H (2.40GHz up to 5.00GHz, 24MB Cache), 14 cores, 20 threads
- GPU: NVIDIA GeForce RTX 4060 8GB GDDR6, TGP: 140W

3. Comparison Result

Metric/Model	ASTGCN (2019)	ASTGNN (2021)	DSTAGNN (2022)
RMSE	21.80	18.44	19.30
MASE	32.82	31.02	31.46

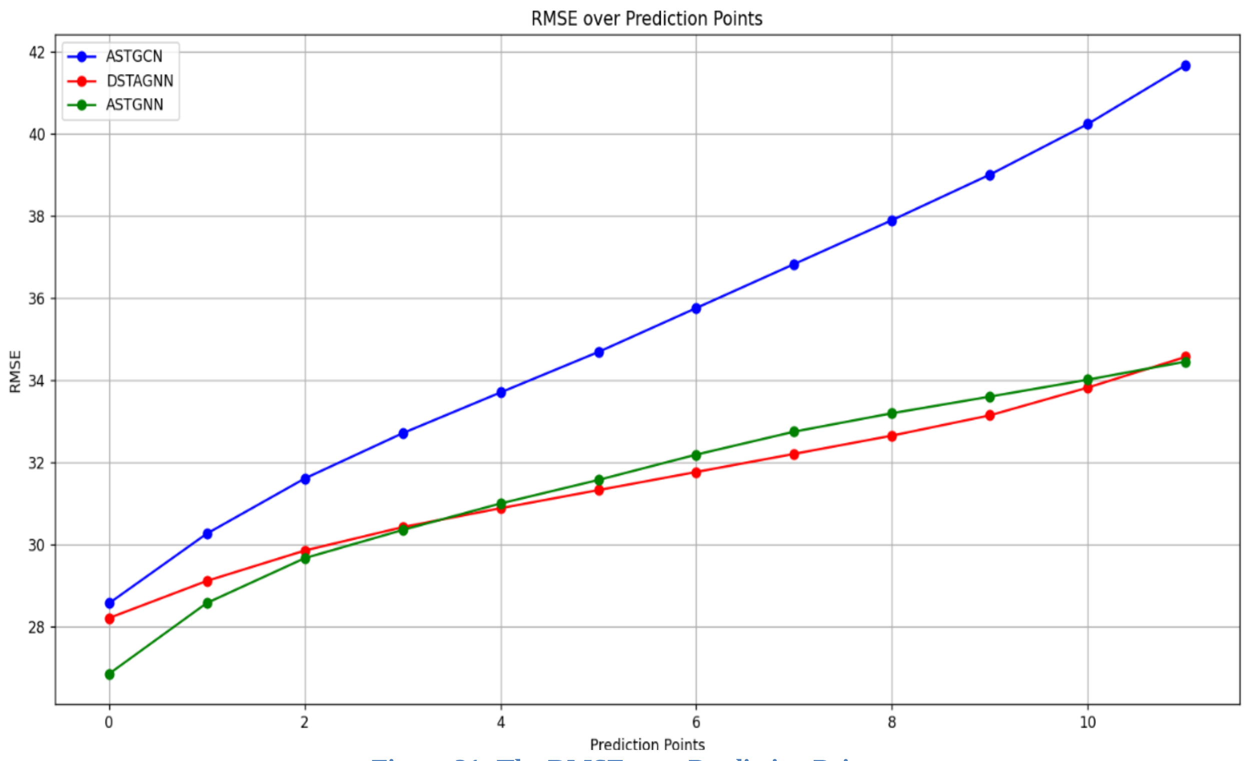


Figure 21: The RMSE over Prediction Points

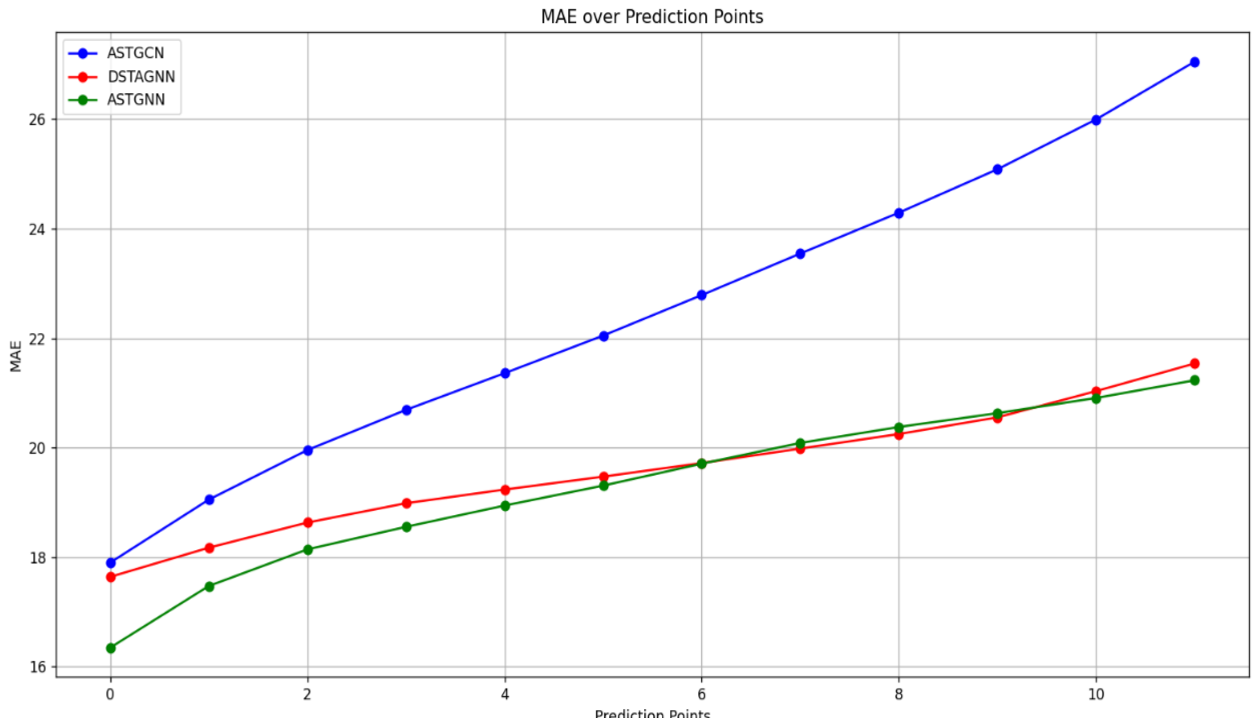


Figure 22: The MAE over Prediction Points

⇒ ASTGCN consistently exhibits the highest error rates among the three which is expected, followed by DSTAGNN, with ASTGNN showing the lowest errors across the prediction points. This suggests that ASTGNN is the most accurate at predicting traffic flow despite coming out a year earlier compared to DSTAGNN, maintaining lower error rates as forecasts extend further into the future.

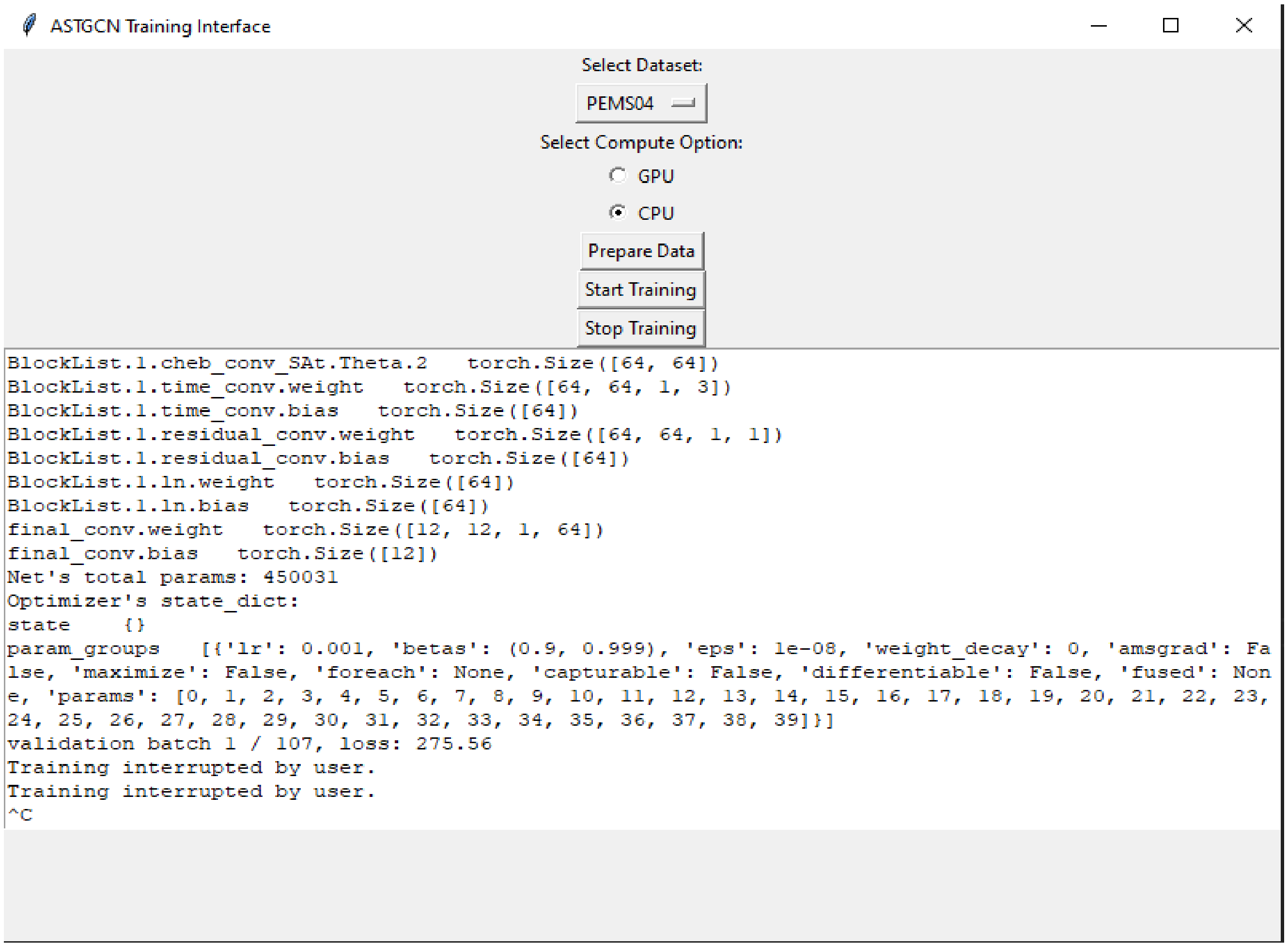
G. DEMONSTRATION

The code uses these external libraries:

- numpy==1.26.4
- torch==2.3.1
- scipy==1.14.0
- matplotlib==3.9.0
- networkx==3.3
- pandas==2.2.2
- tensorflow==2.16.2
- tensorboardX==2.6.2.2
- scikit-learn==1.5.1

We have created a simple GUI for users to run this model from preparing the data to training the model itself using different datasets. The process is structured in a sequence style making the process as simple as possible.

Furthermore, if the prior step is not finished the next step can't happen to ensure minimum error. To open the GUI, run the GUI.py file.



```
BlockList.1.cheb_conv_SAt.Theta.2    torch.Size([64, 64])
BlockList.1.time_conv.weight    torch.Size([64, 64, 1, 3])
BlockList.1.time_conv.bias    torch.Size([64])
BlockList.1.residual_conv.weight    torch.Size([64, 64, 1, 1])
BlockList.1.residual_conv.bias    torch.Size([64])
BlockList.1.ln.weight    torch.Size([64])
BlockList.1.ln.bias    torch.Size([64])
final_conv.weight    torch.Size([12, 12, 1, 64])
final_conv.bias    torch.Size([12])
Net's total params: 450031
Optimizer's state_dict:
state    {}
param_groups    [{'lr': 0.001, 'betas': (0.9, 0.999), 'eps': 1e-08, 'weight_decay': 0, 'amsgrad': Fa
lse, 'maximize': False, 'foreach': None, 'capturable': False, 'differentiable': False, 'fused': Non
e, 'params': [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,
24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39]}]
validation batch 1 / 107, loss: 275.56
Training interrupted by user.
Training interrupted by user.
^C
```

Figure 23: The GUI illustrates the ASTGCN model.

H. EXTENSION

Extension.option2: For the traffic flow prediction system, we use the PemS dataset instead, this dataset as described above contains traffic data collected in the San Francisco Bay Area, using 3848 detectors on 29 roads. This dataset contains the traffic data with dimensions (time steps, sensors, features) for each sensor over time. With the shape of (16992,307,3), there are 16992 time slices with each time slice containing 307 sensors. Each sensor has 3 features, and each time slice is 5 minutes apart. Overall, this dataset is clearly more detailed and more complex to deal with compared to the original dataset that was given to us on Canvas.

I. CONCLUSION

This project study has thoroughly explored the application of the Attention-based Spatial-Temporal Graph Convolutional Network (ASTGCN) model for the prediction of traffic flow dynamics, utilizing the PeMS04 dataset. Our research has demonstrated that ASTGCN, with its innovative integration of spatial-temporal attention mechanisms and graph convolutions, effectively captures the complex, dynamic patterns inherent in traffic data, which are often influenced by both spatial and temporal correlations.

Our comparative analysis with later models like ASTGNN (2021) and DSTAGNN (2022) highlights that while newer models incorporate more refined attention mechanisms and broader data inputs, including external influences like weather and socio-economic factors, ASTGCN remains a foundational approach with robust capabilities in traffic flow forecasting. This is particularly notable in scenarios that demand high responsiveness to spatial and temporal changes without the need for extensive external data inputs.

Researching this topic in the last 3 months of the study period has pushed us to go further into the field of Deep Learning and Neural Networks. To understand ASTGCN, we have spent relentless effort reading cited papers, articles and watching videos on relevant theories present in ASTGCN. Recent problems with our understanding arise every week but we keep pushing, one problem at a time, to have a thorough understanding of each component as well as techniques used in the ASTGCN model, from the dataset to the last prediction step. Overall, we are quite confident in our understanding of the ASTGCN model and its application in real-world traffic prediction scenarios. The firsthand experience has deepened our practical skills and theoretical knowledge, making this project not only a significant academic exercise but also a valuable practical learning experience.

REFERENCES

1. Bruna, J., Zaremba, W., Szlam, A., & Lecun, Y. (2014). Spectral networks and locally connected networks on graphs. In International Conference on Learning Representations.
2. Defferrard, M., Bresson, X., & Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In Advances in Neural Information Processing Systems (pp. 3844-3852).
3. Guo, S., Lin, Y., Feng, N., Song, C., & Wan, H. (2019). Attention Based Spatial-Temporal Graph Convolutional Networks for Traffic Flow Forecasting. Proceedings of the AAAI Conference on Artificial Intelligence, 33, 922–929. https://doi.org/10.1609/aaai.v33i01.3301922
4. Hammond, D. K., Vandergheynst, P., & Gribonval, R. (2011). Wavelets on graphs via spectral graph theory. Applied and Computational Harmonic Analysis, 30(2), 129-150.
5. Lan, S., Ma, Y., Huang, W., Wang, W., Yang, H., & Li, P. (2022). DSTAGNN: Dynamic Spatial-Temporal Aware Graph Neural Network for Traffic Flow Forecasting. Proceedings of the 39th International Conference on Machine Learning, in Proceedings of Machine Learning Research, 162, 11906-11917. https://proceedings.mlr.press/v162/lan22a.html
6. Liang, Y., Ke, S., Zhang, J., Yi, X., & Zheng, Y. (2018). GeoMAN: Multi-level attention networks for geo-sensory time series prediction. In Proceedings of the 27th International Joint Conference on Artificial Intelligence (pp. 3428-3434).
7. Li, Y., Yu, R., Shahabi, C., & Liu, Y. (2018). Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In International Conference on Learning Representations.
8. Guo, S., Lin, Y., Wan, H., Li, X., & Cong, G. (2022). Learning Dynamics and Heterogeneity of Spatial-Temporal Graph Data for Traffic Forecasting. IEEE Transactions on Knowledge and Data Engineering, 34(11), 5415-5428. https://doi.org/10.1109/TKDE.2021.3056502
9. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In Advances in Neural Information Processing Systems (pp. 5998-6008).

10. Zhang, J., Zheng, Y., Qi, D., Li, R., & Yi, X. (2018). DNN-based prediction model for spatio-temporal data. In Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (pp. 92-101).