Predicting Public Transport flow with AI

Forecasting chaotic flows with complex AI architectures

Hephaestus Applied Artificial Intelligence Association

Authors:

Member	Role
Andrea Camicia	Head
Sofia Guidotti	Member
Pavel Siakovpr	Member
Maria Ester Massari	Member

Milan, February 18, 2025

Contents

1 | Mathematical Formulation of the Spatio-Temporal Graph Transformer

In this section, we present an in-depth, mathematically rigorous delineation of the architecture and functional mechanisms embedded in the provided code. This *Spatio-Temporal Transformer* (STGT) is designed to handle relational (graph-based) and temporal data simultaneously, leveraging (i) a learnable temporal encoding scheme, (ii) a spatial attention module endowed with a trainable adjacency matrix, (iii) a temporal attention module, (iv) gated residual connections, and (v) a global node-level multi-head attention that operates on the final latent embeddings.

1.1 | Motivation for Key Design Choices

- **Transformer Paradigm:** Traditional RNN-based models (e.g., LSTM, GRU) can struggle with long-range dependencies in time-series forecasting, especially when the number of nodes N is large. Transformers address this by leveraging attention mechanisms that can capture distant temporal and spatial interactions in parallel.
- **Trainable Adjacency:** While static adjacency structures are informative, many real-world relationships are dynamic or partially unknown. A learned adjacency offset $\mathbf{W}_{\text{adj}}$ allows the model to refine the original graph topology.
- **Gated Residual Connections:** These provide a flexible interpolation between the identity mapping and the transformed representation, which can stabilize training and help the network retain essential information.

1.2 | Notation and Dimensions

- Let B denote the **batch size**, i.e., the number of sequences processed simultaneously.
- Let T represent the **number of time steps** in each input sequence.
- Let N be the **number of nodes** in the underlying graph $\mathcal{G}$.
- Let F signify the **number of raw input features** per node at each time step.
- Let d be the **model dimension**, also referred to as the embedding dimension.

Hence, the input can be conceptualized as a fourth-order tensor

$$\mathbf{X} \in \mathbb{R}^{B\times T\times N\times F},$$

where $\mathbf{X}_{b,\,t,\,n,\,f}$ represents the feature value corresponding to the b-th sample, at time step t, for node n, and feature index f.

1.3 | Time-Learnable Encoding

The model replaces canonical sinusoidal positional encodings with a *trainable* set of embeddings for each possible time index. Concretely, define a learnable parameter matrix

$$\mathbf{E}_{\text{time}} \in \mathbb{R}^{T_{\max}\times d},$$

where $T_{\max}$ is the maximum sequence length the architecture can accommodate (often set large, e.g., 1000). If an input sequence has length $T \le T_{\max}$, we extract

$$\mathbf{E}_{\text{slice}} = \mathbf{E}_{\text{time}}[0 : T-1] \in \mathbb{R}^{T\times d}.$$

We then broadcast this slice along the node dimension and *add* it to a learned linear projection of the raw input. Formally:

$$\mathbf{X}'_{b,\,t,\,n,\,:} = \mathbf{W}_{\text{proj}}\,\mathbf{X}_{b,\,t,\,n,\,:} + \mathbf{b}_{\text{proj}} + \mathbf{E}_{\text{slice}}[t,\,:],$$

where $(\mathbf{W}_{\text{proj}}, \mathbf{b}_{\text{proj}})$ denote the trainable parameters of the initial linear (Dense) layer mapping $\mathbb{R}^F \to \mathbb{R}^d$. This yields a transformed representation

$$\mathbf{X}' \in \mathbb{R}^{B\times T\times N\times d},$$

which explicitly encodes the temporal index in a learnable manner.

1.4 | Graph Adjacency, Trainable Weights, and Spatial Attention

1.4.1 | Base Adjacency and Learnable Matrix

We are given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $|\mathcal{V}| = N$. Let

$$\mathbf{A} \in \{0,1\}^{N \times N}$$

be the static adjacency matrix derived from $\mathcal{G}$, such that $\mathbf{A}_{i,j} = 1$ if and only if an edge exists between node i and node j. The code introduces a trainable real-valued matrix

$$\mathbf{W}_{\text{adj}} \in \mathbb{R}^{N \times N},$$

initialized with small random values (e.g., uniform in $[-0.01, 0.01]$). We combine these via

$$\mathbf{A}_{\text{eff}} = \sigma(\mathbf{A} + \mathbf{W}_{\text{adj}}),$$

where $\sigma(\cdot)$ is the element-wise logistic sigmoid function. This allows the network to deviate from the base adjacency pattern by *reinforcing* or *diminishing* each connection in a continuous manner. The emergent matrix $\mathbf{A}_{\text{eff}}$ thus encodes a *soft connectivity* that is learned end-to-end via gradient-based optimization.

1.4.2 | Spatial Multi-Head Attention

We define $\mathbf{Z} \in \mathbb{R}^{B \times T \times N \times d}$ as the input to the spatial attention mechanism. For multi-head attention (MHA) over the node dimension, we first reshape:

$$\mathbf{Z}_{\text{reshaped}} \in \mathbb{R}^{(B \times T) \times N \times d}.$$

Within a single attention head, recall the classical Transformer attention formulation: let h be the number of heads, and let $d_k = d/h$ be the dimension per head (assuming d is divisible by h). For each head $i \in \{1, 2, \ldots, h\}$, we compute:

$$\mathbf{Q}_i = \mathbf{Z}_{\text{reshaped}}\, \mathbf{W}_i^Q, \quad \mathbf{K}_i = \mathbf{Z}_{\text{reshaped}}\, \mathbf{W}_i^K, \quad \mathbf{V}_i = \mathbf{Z}_{\text{reshaped}}\, \mathbf{W}_i^V,$$

where $\mathbf{W}_i^Q, \mathbf{W}_i^K, \mathbf{W}_i^V \in \mathbb{R}^{d \times d_k}$ are learnable projection matrices. The raw attention output for head i is:

$$\text{Head}_i = \text{softmax}\left(\frac{\mathbf{Q}_i\, \mathbf{K}_i^\top}{\sqrt{d_k}} + \mathbf{M}_{\text{adj}}\right) \mathbf{V}_i,$$

where $\mathbf{M}_{\text{adj}}$ is an attention *mask* derived from $\mathbf{A}_{\text{eff}}$. Specifically, the code sets

$$[\mathbf{M}_{\text{adj}}]_{j,k} = \begin{cases} 0, & \text{if } \mathbf{A}_{\text{eff}}(j,k) \text{ is large (connected)}, \\ -10^9, & \text{if } \mathbf{A}_{\text{eff}}(j,k) \text{ is near 0 (disconnected)}. \end{cases}$$

Hence, the self-attention weights between two nodes j and k become negligible if $\mathbf{A}_{\text{eff}}(j,k) \approx 0$.
The h heads are then concatenated:

$$\text{MHA}(\mathbf{Z}_{\text{reshaped}}) = \text{Concat}(\text{Head}_1, \text{Head}_2, \ldots, \text{Head}_h)\, \mathbf{W}^O,$$

where $\mathbf{W}^O \in \mathbb{R}^{(h \cdot d_k) \times d}$ is a trainable output projection. We then reshape back to $\mathbb{R}^{B \times T \times N \times d}$. Let $\widetilde{\mathbf{Z}}_{\text{spatial}}$ denote this MHA output. The final step is a residual connection plus layer normalization:

$$\mathbf{Z}_{\text{out}}^{(\text{spatial})} = \text{LayerNorm}\Big(\mathbf{Z} + \text{Dropout}(\widetilde{\mathbf{Z}}_{\text{spatial}})\Big).$$

1.5 | Temporal Attention

Given $\mathbf{Z}_{\text{out}}^{(\text{spatial})} \in \mathbb{R}^{B \times T \times N \times d}$, we apply *temporal* multi-head attention. Here, each node's temporal evolution is treated as a standalone sequence, decoupled from other nodes. Formally, for each node n, we have:

$$\mathbf{Z}_{:,:,n,:} \in \mathbb{R}^{B \times T \times d}.$$

We reshape to:

$$\mathbf{Z}_{\text{temp\_reshaped}} \in \mathbb{R}^{(B\times N)\times T\times d},$$

then perform standard multi-head self-attention across the time dimension:

$$\mathbf{Z}_{\text{temp\_attn}} = \text{MHA}\big(\mathbf{Z}_{\text{temp\_reshaped}}, \mathbf{Z}_{\text{temp\_reshaped}}, \mathbf{Z}_{\text{temp\_reshaped}}\big).$$

We map the result back to $\mathbb{R}^{B\times T\times N\times d}$ and again employ a residual+normalization step:

$$\mathbf{Z}_{\text{out}}^{(\text{temporal})} = \text{LayerNorm}\Big(\mathbf{Z}_{\text{out}}^{(\text{spatial})} + \text{Dropout}(\mathbf{Z}_{\text{temp\_attn}})\Big).$$

1.6 | Feed-Forward Network and Gated Residual

Each *STAttentionBlock* concludes with a feed-forward sub-network of dimension d_{ff}, typically realized as a stack of `Dense` layers with nonlinear activations (e.g. `GELU`). Concretely, let $\mathbf{Y} = \mathbf{Z}_{\text{out}}^{(\text{temporal})}$. Then:

$$\mathbf{F}_1 = \text{Dropout}\big(\text{GELU}(\mathbf{Y}\mathbf{W}_1 + \mathbf{b}_1)\big), \quad \mathbf{F}_2 = \text{Dropout}\big(\text{GELU}(\mathbf{F}_1\mathbf{W}_2 + \mathbf{b}_2)\big),$$

and so on, culminating in a projection back to d dimensions:

$$\mathbf{F}_{\text{out}} = \mathbf{F}_2\mathbf{W}_3 + \mathbf{b}_3,$$

where $(\mathbf{W}_1, \mathbf{b}_1)$, $(\mathbf{W}_2, \mathbf{b}_2)$, and $(\mathbf{W}_3, \mathbf{b}_3)$ are trainable parameters. We then incorporate a *gated residual* connection:

$$\mathbf{Z}_{\text{gated}} = \alpha\,\mathbf{Y} \;+\; (1-\alpha)\,\mathbf{F}_{\text{out}},$$

where α is a **trainable scalar** (e.g. initialized to 0.5). Finally, we apply layer normalization:

$$\mathbf{Z}_{\text{block\_out}} = \text{LayerNorm}\Big(\mathbf{Z}_{\text{gated}}\Big).$$

By stacking multiple STAttentionBlocks (indexed by $l = 1, 2, \ldots, L$), we obtain a deeper spatio-temporal feature representation.

1.7 | Global Node-Level Attention

After the final block, we have $\mathbf{Z}_{\text{block\_out}} \in \mathbb{R}^{B\times T\times N\times d}$. We *select the last time step*, $t = T-1$:

$$\mathbf{H} = \mathbf{Z}_{\text{block\_out}}[:, T-1, :, :] \;\in\; \mathbb{R}^{B\times N\times d}.$$

This $\mathbf{H}$ is a sequence of N node embeddings, each in $\mathbb{R}^d$. We then apply another multi-head attention block:

$$\mathbf{H}_{\text{attn}} = \text{MHA}\big(\mathbf{H}, \mathbf{H}, \mathbf{H}\big),$$

with, for example, 4 heads. A residual connection plus layer normalization yields

$$\mathbf{H}_{\text{out}} = \text{LayerNorm}\Big(\mathbf{H} + \mathbf{H}_{\text{attn}}\Big).$$

1.8 | Final Node-Wise Predictions

We next pass each node's representation $\mathbf{H}_{\text{out}} \in \mathbb{R}^{B\times N\times d}$ through a small MLP, typically consisting of one or two `Dense` layers with `GELU` activations, plus dropout. Denoting these transformations collectively as $\Theta(\cdot)$, we obtain:

$$\mathbf{R} = \Theta\big(\mathbf{H}_{\text{out}}\big) \;\in\; \mathbb{R}^{B\times N\times d'},$$

where d' is an intermediate hidden size (e.g. 128 or 256). The final output layer, Dense(1), yields

$$\widehat{\mathbf{Y}} \;=\; \text{Dense}(\mathbf{R}) \;\in\; \mathbb{R}^{B\times N\times 1},$$

providing one scalar regression output per node.

1.9 | Objective, R^2 Metric, and Training

While the code snippet accommodates a generic loss function (e.g. MSE), we also define a custom R^2 metric:

$$R^2(\mathbf{Y}, \widehat{\mathbf{Y}}) \;=\; 1 \;-\; \frac{\sum_{b,n} \big(Y_{b,n} - \widehat{Y}_{b,n}\big)^2}{\sum_{b,n} \big(Y_{b,n} - \overline{Y}\big)^2},$$

where $\overline{Y}$ is the mean of all ground-truth values $\mathbf{Y}_{b,n}$ in the training (or validation) set. At each iteration of training (e.g., SGD or Adam), the model updates:

- The time-embedding weights $\mathbf{E}_{\text{time}}$,
- The adjacency offsets $\mathbf{W}_{\text{adj}}$,
- All parameters in the multi-head attention layers ($\mathbf{W}_i^Q, \mathbf{W}_i^K, \mathbf{W}_i^V, \mathbf{W}^O$, etc.),
- The feed-forward networks and gating parameters (α),
- The final MLP parameters for the node-wise predictions,

in order to minimize the chosen loss (e.g. MSE) and potentially maximize R^2 on validation sets.

1.10 | Overall Architectural Summary

To recapitulate in a concise blueprint:

1. **Time Embedding:** $\mathbf{X}' = \text{TimeLearnableEncoding}(\mathbf{X}) \in \mathbb{R}^{B\times T\times N\times d}$.
2. **STAttention Blocks (stacked L times):** for $l \in \{1, \ldots, L\}$,

$$\mathbf{Z}^{(l)} = \text{LayerNorm}\Big(\text{GatedRes}\big(\text{FFN}(\text{TemporalAttn}(\text{SpatialAttn}(\mathbf{Z}^{(l-1)}))))\Big).$$

   Each SpatialAttn uses $\mathbf{A}_{\text{eff}} = \sigma(\mathbf{A} + \mathbf{W}_{\text{adj}})$.
3. **Extraction of Final Time Step:** $\mathbf{H} = \mathbf{Z}^{(L)}[:, T-1, :, :]$.
4. **Global Node-Level Attention:** $\mathbf{H}_{\text{out}} = \text{LayerNorm}\Big(\mathbf{H} + \text{MHA}(\mathbf{H}, \mathbf{H}, \mathbf{H})\Big)$.
5. **Projection to Scalar Output:** $\widehat{\mathbf{Y}} \in \mathbb{R}^{B\times N\times 1} = \text{Dense}\big(\text{MLP}(\mathbf{H}_{\text{out}})\big)$.

2 | Data Generation and Preprocessing

To evaluate and illustrate the Spatio-Temporal Transformer, we employ a synthetic transport dataset that preserves a number of realistic characteristics: demand surges at peak times, seasonal fluctuations, and event-induced irregularities. The Python functions `generate_transport_data` and `preprocess_data` encapsulate this logic. We now provide an overview of how the data is generated and normalized before being fed into the model.

2.1 | Synthetic Data Generation

We generate a date range from a specified start date to an end date at 30-minute intervals. Each node (e.g., a station) is associated with a baseline demand ranging from 100 to 1000 passengers. We introduce:

- **Weekday vs. Weekend Adjustments:** A factor to reduce demand on weekends ($\sim$0.8).
- **Peak Hour Multipliers:** Increased values (up to 1.5$\times$) for morning and evening rush hours.
- **Seasonal Sinusoidal Factors:** A 20% periodic fluctuation across the year.
- **Noise and Random Variation:** Gaussian noise ($\mathcal{N}(1, 0.05)$) and minor random scalings (uniform in $[0.9, 1.1]$) to simulate fine-grained fluctuations.
- **Temporal Trend:** A subtle upward drift introduced over time.
- **Events (Optional):** Further multipliers for dates that fall within special events.

After constructing the full X (with dimensions [TimeSteps] $\times$ [Nodes] $\times$ [Features]), we produce input-output sequences by choosing a history length (`time_steps`). Each sample thus consists of a window of $\mathbf{X}$ spanning `time_steps` intervals, and we define the target y via a simple combination of the features (e.g., $\sqrt{X_i}$, $\log(1 + X_i)$, etc.) plus a portion of the adjacency-weighted average. This encourages the model to capture both node-specific behaviors and relational influences.

2.2 | Preprocessing and Splitting

We apply a MinMax scaling procedure separately to the features X and the target y. By default, we split the data into a training set (first 80% of available years) and a test set (remaining 20%). The final output is:

- $\mathbf{X}_{\text{train}}, \mathbf{y}_{\text{train}}$ and $\mathbf{X}_{\text{test}}, \mathbf{y}_{\text{test}}$ in normalized form,
- The corresponding scaler_X and scaler_y objects to transform new data or invert the normalization,
- The *years* and *dates* arrays to keep track of each sequence's temporal context.

Through these procedures, the network is presented with input sequences capturing multi-featured node activity over a temporal window, enabling robust modeling of complex spatio-temporal patterns.

3 | Proposed Model Architecture Diagram

Figure 3.2 shows a high-level sketch of the Spatio-Temporal Graph Transformer (STGT) with the major components discussed in Section 1. Notably, each node's historical data is embedded (via time-learnable encodings), then passed through spatial and temporal multi-head attention blocks, followed by a final global node-level attention module.

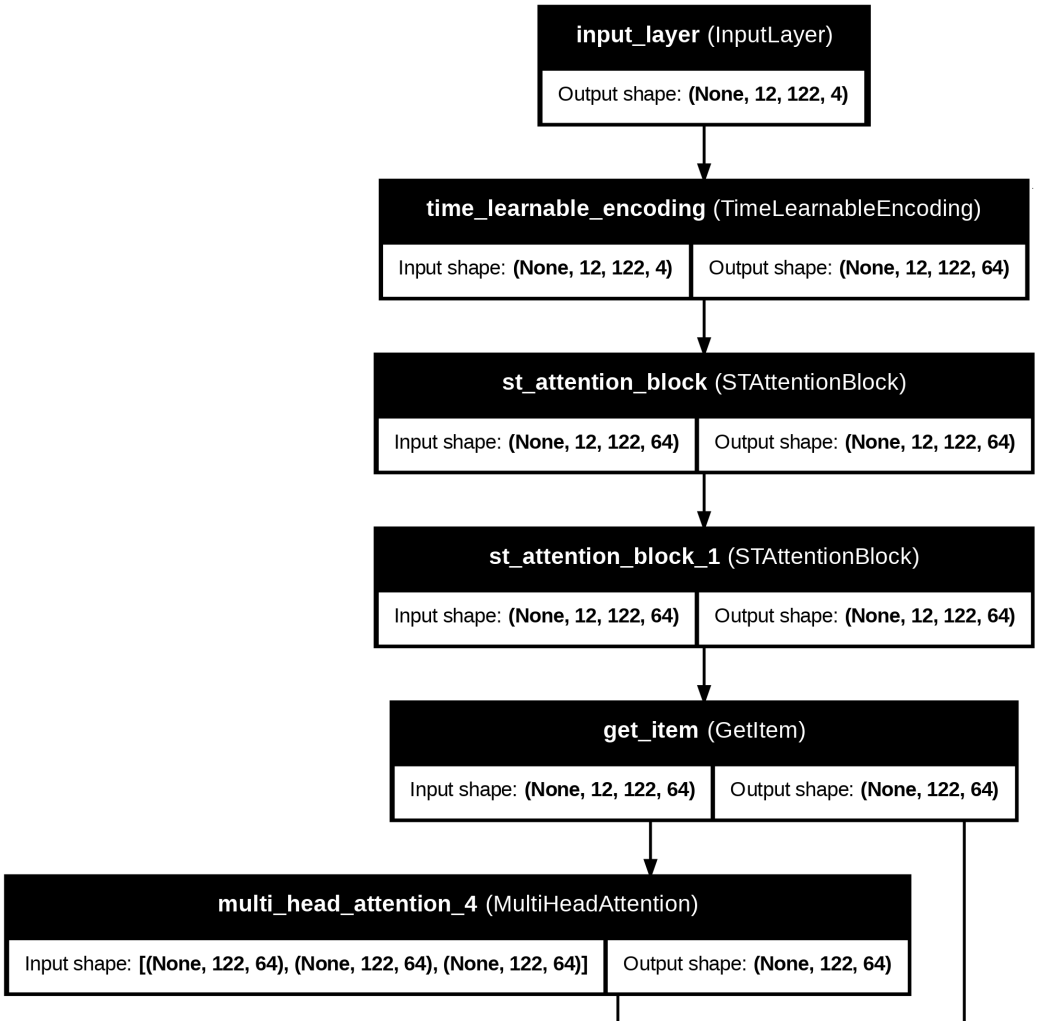

Figure 3.1: High-level architecture of the Spatio-Temporal Graph Transformer (STGT). Input data is first projected and combined with a trainable temporal embedding. The network then applies repeated blocks of *spatial attention* and *temporal attention* (with a learnable adjacency), culminating in a final node-level attention and an MLP-based output.

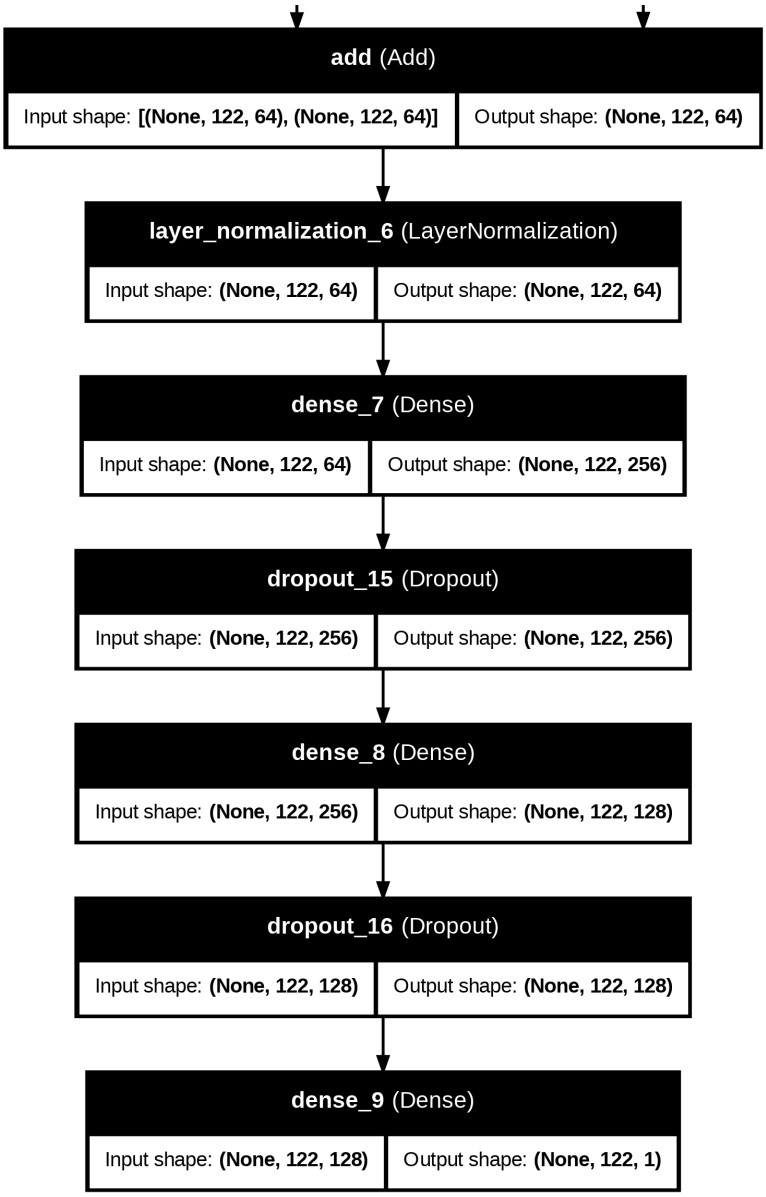

Figure 3.2: Continuation of 3.1.

Each multi-head attention block operates either on the node dimension (spatial) or on the time dimension (temporal). The adjacency offsets $\mathbf{W}_{\text{adj}}$ are learned end-to-end, allowing the model to refine or discover node-to-node relationships. A final MLP yields one scalar prediction per node. Gated residual connections (see Section 1.5) help stabilize training by controlling the balance between direct skip-connections and transformed representations.

4 | Experimental Results on Synthetic Data

In this section, we provide a discussion of the model's performance on a purely synthetic dataset. Real-world data were unavailable at the time of this study, so we generated a custom synthetic dataset that reflects **several realistic features**, including:

- **Baseline Time Series:** A smooth periodic signal to model daily or weekly cycles.
- **Gaussian Noise:** Added to each node's time series to capture random fluctuations.
- **Weather Features:** Simulated attributes such as temperature or precipitation, randomly varying but correlated with certain seasonal patterns.
- **Event Features:** Synthetic markers for special events (e.g., holidays), randomly distributed throughout the timeline but influencing spikes in flow.

By incorporating these elements, we aimed to mirror some complexities found in real public transport data—namely periodic behavior, abrupt changes, and correlated external factors.

4.1 | Training Setup and Data Splits

Data Construction. We synthesized a dataset of N nodes, each having F features (including weather, events, and baseline signals). The adjacency matrix $\mathbf{A}$ was chosen to reflect partially connected subgraphs, and small random offsets were introduced.

Train/Test Splits.

- **Training Sequences:** $t = 0$ to $t = 250$ and $t = 251$ to $t = 500$.
- **Long-Horizon Prediction:** The model was tested on forecast windows extending from $t = 12{,}251$ to $t = 12{,}500$ to evaluate long-range predictive capability.

The model was trained for approximately 30–40 epochs. Despite this short regime, the STGT demonstrated quick convergence on the synthetic data, highlighting its capacity to capture both the temporal cycles and the inter-node relationships.

4.2 | Performance Observations and Generalization

Loss Convergence. During training, the mean squared error (MSE) dropped substantially within the first few epochs, suggesting that the network effectively captures the synthetic data's salient features. Additional epochs yield minor improvements, indicating a relatively stable training process.

Cumulative Prediction. To gauge the model's global performance, we aggregate predictions over all nodes at each time step:

$$\widehat{y}_{\text{cumulative}}(t) = \frac{1}{N}\sum_{n=1}^{N}\widehat{Y}_{n,t}, \quad y_{\text{cumulative}}(t) = \frac{1}{N}\sum_{n=1}^{N}Y_{n,t}.$$

This smooths out local node-level fluctuations and emphasizes the general trend. Figure 4.2 illustrates that the predicted and true mean flow values are closely aligned, confirming the STGT's ability to reproduce global behaviors.

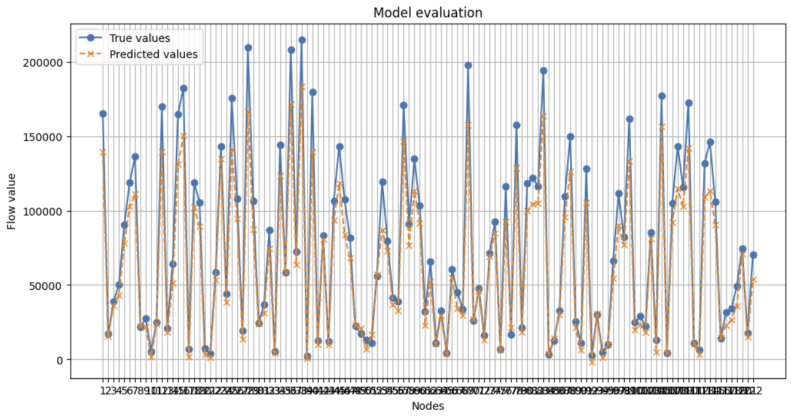

Figure 4.1: Comparison of cumulative (mean) true vs. predicted flow values across all nodes.

Long-Horizon Forecasts. The model's extrapolation to $t \approx 12{,}500$ reproduced the cyclic patterns induced by the synthetic daily or weekly cycles, although certain outliers (e.g., event-driven spikes) were challenging. Overall, the ability to track the general cyclical behavior over such an extended horizon is encouraging.

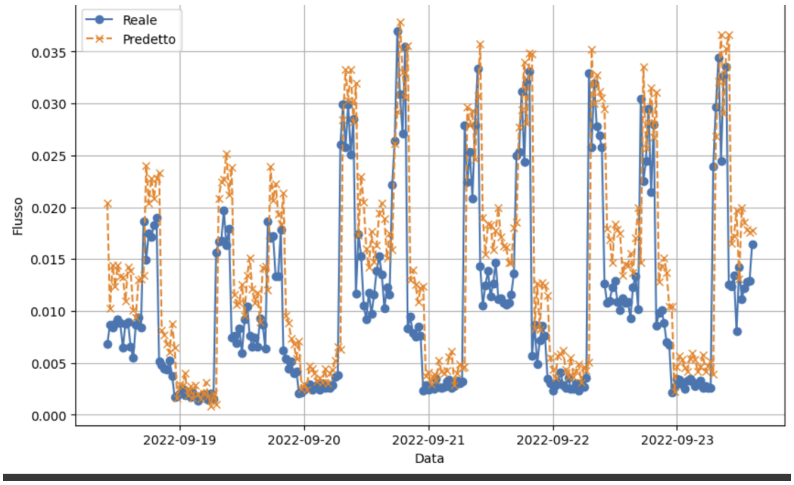

Figure 4.2: Comparison of true vs. predicted flow value at one specific node of the network.

Adjacency Adaptation. Even in this synthetic scenario, the trainable adjacency offsets $\mathbf{W}_{\text{adj}}$ proved beneficial. Nodes with correlated signals were reinforced, while weak or spurious connections tended toward near-zero effective adjacency. This aligns well with the intuition that the model can "re-wire" the graph if the original topology is incomplete or partially incorrect.

5 | Computational Complexity and Scalability

Although the emphasis here is on modeling efficacy rather than large-scale benchmarks, it is important to recognize the theoretical complexity of STGT layers.

5.1 | Spatial and Temporal Attention Complexities

Spatial Attention. For N nodes, naive self-attention is $\mathcal{O}(N^2 d)$ per head. Since this is applied for each time step and batch, memory and compute can become substantial as N grows.

Temporal Attention. For each node, attending over T time steps yields $\mathcal{O}(T^2 d)$. Multiplied by N nodes and B samples, this can also grow quickly.

5.2 | Future Efficiency Strategies

For very large N or T, one may consider:

- **Sparse Attention:** Apply block- or locality-based attention patterns.
- **Low-Rank Factorization:** Decompose large attention matrices to reduce complexity.
- **Distributed Training:** Split data or model parameters across multiple GPUs/TPUs.

6 | Theoretical Justification and Interpretability

6.1 | Why Transformers for Spatio-Temporal Data?

- **Parallel Computation:** Unlike RNNs, Transformers process entire sequences in parallel.
- **Adaptive Focus:** Attention heads learn which nodes and time steps deserve the most "focus."
- **Learnable Graph Structure:** Through $\mathbf{W}_{\text{adj}}$, the model can discover latent links that are not obvious in an a priori adjacency matrix.

6.2 | Attention Weights and Learned Adjacency

Visualizing attention weights provides a window into how the network redistributes emphasis across nodes and time steps. In practice, one can:

- Plot heatmaps of $\sigma(\mathbf{A} + \mathbf{W}_{\text{adj}})$ to reveal newly "discovered" or diminished connections.
- Examine the final node-level attention to see which nodes have the greatest pairwise influence.

7 | Discussion of Limitations and Future Work

While the synthetic experiments demonstrate the feasibility of STGT for complex spatio-temporal prediction, the following points warrant further investigation:

7.1 | Synthetic Data vs. Real-World Generalization

Our results currently rely on synthetic data, which, despite efforts to include noise, weather, and event features, cannot fully replicate the richness of real operational environments. Future work should apply STGT to real transport datasets, addressing:

- **Missing Data and Irregular Sampling**
- **Sensor or Logging Errors**
- **Varied Graph Topologies**

7.2 | Model Complexity

- **Memory Footprint:** Both spatial and temporal attention are $\mathcal{O}(N^2)$ and $\mathcal{O}(T^2)$, respectively.
- **Scalability:** For $N \gg 10^3$ or $T \gg 10^3$, specialized attention mechanisms or distributed training may be required.

7.3 | Improved Interpretability

Additional methods (e.g., layerwise relevance propagation, attention roll-out) could help domain experts understand why certain nodes are prioritized or how the adjacency matrix evolves during training.

7.4 | Conclusion

In summary, the Spatio-Temporal Transformer architecture described here—featuring learnable temporal embeddings, spatial attention with adjustable adjacency, temporal attention for each node sequence, and a gated residual feed-forward network—provides a compelling framework for high-dimensional graph-based time series forecasting. While these initial results using synthetic data are promising, real-world validation and further research on scalability, interpretability, and model optimization remain essential next steps in advancing this methodology.