An-Najah National University

Faculty of Graduate Studies

A HYBRID DEEP LEARNING MODEL FOR FORECASTING $PM_{2.5}$ AIR POLLUTANT CONCENTRATIONS

By

Asma Mousa (Mohammad Ali) Massad

Supervisors

Dr. Anas Toma

Dr. Abdelhaleem Khader

This Thesis is Submitted in Partial Fulfillment of the Requirements for the Degree of Master in Artificial Intelligence, Faculty of Graduate Studies, An-Najah National University, Nablus - Palestine.

2024

A HYBRID DEEP LEARNING MODEL FOR FORECASTING $PM_{2.5}$ AIR POLLUTANT CONCENTRATIONS

By

Asma Mousa (Mohammad Ali) Massad

This Thesis was Defended Successfully on 18/12/2024 and approved by

Dr. Anas Toma	
Supervisor	Signature
Dr. Abdelhaleem Khader	
Co-Supervisor	Signature
Dr. Ahmad Hasasneh	
External Examiner	Signature
Dr. Hamed Abdelhaq	
Internal Examiner	Signature

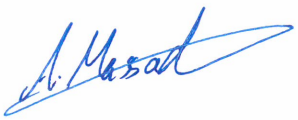
Declaration

I, the undersigned, declare that I submitted the thesis entitled:

A HYBRID DEEP LEARNING MODEL FOR $PM_{2.5}$ FORECASTING

I declare that the work provided in this thesis, unless otherwise referenced, is the researcher's own work, and has not been submitted elsewhere for any other degree or qualification.

Student's Name: **Asma Mousa 'Mohammad Ali' Massad**

Signature: 

Date: **18/12/2024**

List of Contents

List of Tables

List of Figures

List of Appendices

A HYBRID DEEP LEARNING MODEL FOR FORECASTING $PM_{2.5}$ AIR POLLUTANT CONCENTRATIONS

By
Asma Mousa (Mohammad Ali) Massad
Supervisors
Dr. Anas Toma
Dr. Abdelhaleem Khader

Abstract

Air quality forecasting is a crucial research field that aids scientists and policymakers in making informed decisions to combat air pollution. Among various pollutants, $PM_{2.5}$ - particulate matter with a diameter smaller than 2.5 micrometers- poses significant health risks, as it can reach the lower respiratory tract and enter the bloodstream. Accurately forecasting $PM_{2.5}$ levels is thus essential. Although machine learning-based spatiotemporal forecasting models have advanced, the pursuit for more accurate forecasts continues. The use of hybrid deep learning models for $PM_{2.5}$ forecasting represents a promising and active area of research, as these models aim to capture complex spatiotemporal dependencies more effectively.

We developed a Dynamic Graph Attention Network (DyGAT) to model spatial dependencies effectively. DyGAT leverages engineered edge features, including distance, wind speed, and wind direction, while using attention mechanisms to capture the dynamic nature of these dependencies. DyGAT was then combined with Informer, a Transformer for efficient time-series forecasting, to capture spatial and temporal patterns comprehensively, improving prediction accuracy. Our model was evaluated on a benchmark dataset from Beijing, with 420,768 records over four years. DyGAT-Informer outperformed a version without the DyGAT component and other baseline models. It achieved 50.43 for MAE, 79.9 for RMSE and 28.88% for SMAPE, compared to 51.44 for MAE, 80.83 for RMSE and 30.25% for SMAPE in the next best model.

Additionally, we conducted a case study using a dataset from Nablus, Palestine, consisting of 2692 records per station over a two months period. We incorporated geospatial features about nearby pollution sources into the dataset. Due to the insufficient number of records in the Nablus dataset for training the Informer, it was replaced with a sequence-to-sequence Long Short-Term Memory (LSTM) model. DyGAT-LSTM,

trained with additional geospatial features about nearby pollution sources, achieved a 2.08% reduction in MAE, 1.17% in RMSE, and 1.96% in SMAPE. This confirms the benefit of incorporating such data. Finally, despite the short distances between stations, DyGAT successfully captured spatial dependencies, where DyGAT-LSTM achieved a reduction of 3.13% in MAE, 1.48% in RMSE, and 3.67% in SMAPE when compared to the LSTM-only model.

Keywords: Spatiotemporal Forecasting, Air Quality, $PM_{2.5}$ Forecasting, Hybrid Deep Learning Model, Graph Attention Networks, Transformers.

Chapter One
Introduction

Air pollution is one of the existential threats to humans in our modern societies. According to the World Health Organization (WHO) in 2019, 99% of the world's population live in places where air pollution levels are above the WHO air quality guidelines [1]. There are many types of air pollutants, the most common ones are Particulate Matter (PM), Carbon monoxide (CO), Ozone (O3), Sulphur dioxide (SO2) and Nitrogen dioxide (NO2). Each one of these pollutants has its own effects on the population's health and other environmental issues [2]. In general, air pollutants can affect different systems and organs in the human body; they can cause cancers (particularly lung cancer) and they are correlated with decreased cancer survival rates [3]. Additionally, exposure to air pollutants is linked to chronic respiratory and cardiovascular diseases [4]. Air pollution is linked to one in nine global deaths and seven million annual premature deaths [5].

Particulate Matter (PM) is one of the prominent air pollutants that directly affect the population's health [6]. PM it is a generic term to describe a variety of pollutants suspended in the air that may differ in their chemical and physical properties. PM particles are usually classified according to their aerodynamic diameter; the two common classes are PM_{10} and $PM_{2.5}$. PM10 refers to particles that have an aerodynamic diameter less than 10 µm, while $PM_{2.5}$ refers to particles with aerodynamic diameter less than 2.5 µm [2]. While all classes of PM pollutants can pose health risks to the human body, finer particles like $PM_{2.5}$ are more hazardous [4]. Exposure to fine particles like $PM_{2.5}$ worsens illnesses like asthma, cancer, and diabetes, while also harming mental health and cognitive development [5].

General examples of air pollutants sources are industrial emissions, fuel combustion, and photochemical reactions from plants. Agricultural and domestic use of herbicides, insecticides, are another source of air pollutants [2]. To deal with the hazards of air pollutants like $PM_{2.5}$, governments and policy makers need models that can predict the concentration levels of these pollutants in order to plan future environmental policies and issue alerts to the population when $PM_{2.5}$ levels are high.

1.1 Theoretical Basis

Forecasting $PM_{2.5}$ concentrations is crucial due to the significant health risks associated with fine particulate matter exposure. Accurate forecasts enable timely public health warnings, allowing individuals to take preventive measures during high pollution periods. Moreover, reliable forecasts assist policymakers in implementing effective air pollution control strategies, thereby improving overall air quality [7]. Additionally, air pollution control strategies are typically long-term and costly. Therefore, using air pollutants forecasting models, specifically for $PM_{2.5}$, can assess the effectiveness of these measures by modeling trends and investigating the impact of applied interventions and air pollution control strategies [8].

$PM_{2.5}$ forecasting models could be divided into traditional approaches and Machine Learning (ML) based approaches. Among traditional approaches, the most commonly used ones are Chemical Transport Models (CTMs) and statistical models. However, these approaches face many difficulties like the complexity of atmospheric processes over simplification of the underlying processes that produce $PM_{2.5}$, the lack of adaptability – since these models are usually built for a specific region – and finally, their inability to capture the nonlinear behavior and interactions between the air components [9].

Due to recent advancement in relatively cheap and reliable measurement instruments, a large amount of historical air quality data became available. Additionally, the computation power has increased rapidly in the last few years, which encouraged researchers to explore a variety of machine learning based architectures that utilize historical data to overcome some of the limitations of traditional methods like modeling nonlinear relations and over simplification of the processes that happen in the atmosphere and affect the air quality [9] [10].

1.2 Related work

Machine learning models are becoming widely used tools for researchers to predict the air quality index and the concentrations of specific pollutants, such as $PM_{2.5}$. A large variety of traditional machine learning algorithms and deep learning models were built to forecast air quality and pollutants concentrations, each model has its own features, strengths and weakness. The choice of a forecasting model depends on the available data and purpose of the research [7] [9].

Machine learning approaches are divided into traditional ML and Deep Learning (DL) models. A recent survey [7] that reviewed machine learning models for air quality forecasting for the last ten years found that both traditional ML and DL models are used to build these models, however, deep learning based models are becoming more prominent in recent studies. Deep learning models are capable of modeling high dimensional data and nonlinear relations [9]. The DL models used in the literature for $PM_{2.5}$ forecasting were either a variation of a single DL model or hybrid models that utilize more than one model to build a forecasting model that could deal with different aspects of the data [7].

For traditional ML models, Support Vector Machine (SVM) [11] and Random Forests (RF) [12] were the two most used algorithms. When it comes to DL, the most common models are Multi-Layer Perceptron (MLP) and Long Short-Term Memory (LSTM) neural networks, which is a variant of Recurrent Neural Network (RNN), followed by Convolutional Neural Network (CNN) [7].

Yao et al. [13] used Artificial Neural Network (ANN) to forecast daily $PM_{2.5}$ levels, they used ground monitoring data and incorporated other sources like satellite images, they compared their model with traditional multiple regression models and their predictions were more accurate. Agarwal et al. [14] developed an ANN based model to forecast many air pollutants including $PM_{2.5}$ and PM_{10}, they provided daily forecasts and longer periods forecasts that could extend to four days. Their data had meteorological features and hourly pollution levels, and their model included a dynamic real time correction feature that can correct current predictions dynamically according to the model's performance in previous days.

The previous two studies [13][14] primarily focused on improving the prediction accuracy of traditional MLP networks without introducing significant changes to the architecture of the networks themselves. In contrast, other studies introduced more novel modifications to the MLP architecture to address different aspects in the forecasting process, or achieve specific goals. One study [15] presented an improvement to the performance of MLP by incorporating Kalman filter to the learning algorithm to adapt to the time variation of the air quality system. Another work [16] added a rolling mechanism and a gray model, which was used to preprocess the meteorological data in order to reduce

complexity. Other modifications included a hybrid model combining MLP and linear regression [17] for improving the accuracy of forecasting $PM_{2.5}$ concentrations. In their model the regression model enhances the reliability of the predictions by correcting bias in the neural network's output.

LSTM is another widely used method for $PM_{2.5}$ forecasting due to its ability to handle time series data effectively. Zhou et al. [18] experimented with different variations of LSTM including a shallow and deep LSTM, while another work [19] built a hybrid LSTM-Kalman model, which performed better than the classical LSTM. One study [20] built a model based on LSTM, which they called Multi-output and Multi-index of Supervised Learning (MMSL); it was a spatiotemporal model where they tried to build a prediction model for one location by incorporating the data from other neighboring measuring stations.

One variation of LSTM is Bidirectional LSTM (Bi-LSTM), which keeps information from the past and the future. Madaan et al. [21] built a Bi-LSTM based model with adaptive attention mechanism. Other works have used transfer learning with Bi-LSTM. For example, one study [22] used transfer learning with Bi-LSTM to transfer the knowledge learned within small temporal resolutions into a model that works with larger temporal resolutions. Another study [23] used transfer learning with Bi-LSTM to predict $PM_{2.5}$ at new stations that do not have data. Bi-LSTM was also used in spatiotemporal models [24].

Zhang et al. [25] created a Bi-LSTM with Empirical Mode Decomposition (EMD) model to predict $PM_{2.5}$ values. In their model, they only used historical $PM_{2.5}$ data without any other meteorological data, where historical $PM_{2.5}$ readings were considered as an input signal and the EMD was used as semi-supervised learning algorithm that extracts the hidden frequency features. Their model was developed to enhance the short-term predictions especially when sudden changes are present.

Other studies used sequence-to-sequence (Seq2Seq) architectures, where both the input and output are sequences, Encoder-Decoder models are an example of Seq2Seq models. One study [26] built an Encoder-Decoder model with LSTM, their model showed significant results for long-term forecasting of $PM_{2.5}$. Another study [27] utilized the

Encoder-Decoder for effective $PM_{2.5}$ prediction along with Genetic Algorithm for feature selection and outlier removal to enhance the forecasting accuracy.

Researchers also experimented with hybrid deep learning models to overcome some of the single algorithm models' limitations. These hybrid models became more popular recently due to the rapid increase in computational power. One popular hybrid model is a combination between LSTM and CNN, in some studies [28] [29] that developed hybrid CNN-LSTM, the CNN was used to extract features related to the air quality while the LSTM is used to model the historical process of the time-series data. While Le et al. [30] developed a CNN-LSTM model where they used the combination to manipulate the spatial and temporal features of their data, which included traffic volume data along the $PM_{2.5}$ and meteorological data.

Zhang et al. [31] built a hybrid model that combines Variational Mode Decomposition (VMD) with Bi-LSTM. The VMD was used to decompose the original time series data signal into multiple sub-signal in the frequency domain, their work had better performance than models that used EMD for signal decomposition.

Another way to utilize hybrid models is to build powerful spatiotemporal models by using Graph Neural Networks (GNN) to model the spatial relations, along with another deep learning model to deal with the temporal dependency in the data. Using LSTMs with GNNs is a popular combination to create hybrid spatiotemporal forecasting models.

Some studies used a variant of GNN called Graph Convolutional Network (GCN) with LSTM. Qi et.al. [32] employed a GCN to capture spatial dependencies in the data and LSTM network to model temporal dependencies, and the final forecasts are generated by passing the output through a Fully Connected Network (FCN). To construct the weighted adjacency matrix, they used a formula that calculates the spatial distance between stations. In their approach, nodes are considered connected only if the distance is within 200 km; otherwise, the adjacency matrix entry is set to zero. Teng et.al. [33] used GCN-LSTM architecture similar to the previous study, however they incorporated Aerosol Optical Depth (AOD) data to improve the accuracy of $PM_{2.5}$ forecasts. AOD is a measure of how much aerosol is present in a column of the atmosphere. Another study [34] created a hybrid model using GCN with self-loops and an LSTM with temporal sliding window, to forecast multiple air pollutants. The temporal sliding window moves over the time-

series data to generate overlapping sequences, which are then used to train the LSTM network.

The previous studies [32], [33], [34] used sequential architecture, where the output of GCN is used as input to the LSTM, however a study by Gao et. al. [35] used a parallel integration method between GCNs and Bi-LSTM, where the outputs of the two models are concatenated and passed to a FCN to create forecasts. In addition, one study [36] used GCN with Gated Recurrent Network (GRU) which is another RNN variant. Some studies [37] used another GNN variant called Graph Attention Network (GAT) [38] with LSTM, and another one combined GAT with GRU [39].

Zhou et. al. [40] introduced another GNN based hybrid model, where they used GCN for spatial dependencies and temporal convolution to obtain temporal features, their model used wind-field diffusion distance to describe the relation between each pair of nodes instead of typical Euclidian distance. There are other deep learning algorithms that were used to build $PM_{2.5}$ forecasting models, such as the Deep Belief Network (DBN) [41] and Autoencoder Neural Network [42].

After the introduction of Transformers [43], many researchers started experimenting with variations of Transformer architectures and hybrid models that include them. Liang et al. [44] built a Transformer based $PM_{2.5}$ prediction model and according to their results, the model was able to predict air quality with fine spatial granularity that was not achieved before.

Al-qaness et. al. [45] built 'ResInformer' a model with the Informer [46], which is a Transformer variant, introduced to improve the inference speed of long-sequence predictions. Their work improved the attention distillation block in the Informer. Ma et. al. [47] used the Informer in a spatiotemporal $PM_{2.5}$ forecasting model; the key innovation is adding a spatiotemporal embedding layer to the Informer to model the spatial dependencies in the data. MSAFormer [48] is another Transformer based $PM_{2.5}$ forecasting model, the model uses Transformer with Sparse Autoencoding extracts the most important features from the vast amount of multi-site meteorological data, focusing on the information most relevant to $PM_{2.5}$ prediction. Zhang et al. [49] built an Encoder-Decoder $PM_{2.5}$ prediction model with Sparse attention-based Transformer Networks (STN), they used the sparse attention approach to reduce the time complexity, their results

shows that the model has a small time complexity and has outperformed state-of-the-art models.

Other Studies used Transformers in hybrid models, one study [50] used Informer with GCN, their aim was to improve air quality forecasting by capturing the dynamic and intricate relationships between air pollutants and their environment. Graph Transformers is another hybrid concept that was introduced in the literature [51], this hybrid model was used for many tasks including time-series forecasting tasks. Li et. al. [52] introduced 'Forecaster' which is spatiotemporal forecasting model based on graph Transformers. Graph Transformers were also used in building $PM_{2.5}$ forecasting models. One study [53] introduced Temporal Difference based Graph Transformer Networks (TDGTN) they utilize temporal difference techniques to learn long-term dependencies in $PM_{2.5}$ concentration data.

Although spatiotemporal forecasting models have recently been investigated and used to forecast $PM_{2.5}$ concentrations, many existing approaches do not fully address the dynamic nature of spatial relationships among different locations. Traditional models and hybrid models have demonstrated success in capturing time-dependent patterns and spatial correlations. However, they often rely on static spatial representations, usually governed by the geolocation of the monitoring stations, which overlook the fact that spatial dependencies between stations can change over time due to factors like weather conditions that affect the dispersion process of pollutants such as $PM_{2.5}$. Moreover, while some models do integrate meteorological data into spatial dependency modeling, they typically do not account for the directionality of pollutant dispersion between each pair of monitoring stations.

1.3 Problem Statement and Study Objectives

Despite recent advancements in $PM_{2.5}$ forecasting models, there is still significant potential to improve how these models capture the complex and non-linear spatiotemporal patterns of $PM_{2.5}$ concentrations. Moreover, the integration of geospatial data, such as geographical information about pollution sources into deep learning models remains an underexplored area.

The primary objective of this study is to develop a hybrid spatiotemporal $PM_{2.5}$ forecasting model that addresses the dynamic spatial dependencies between locations and incorporates domain knowledge about pollutant dispersion. This hybrid model consists of two parts: a dynamic variant of GAT that we have developed to capture the dynamic spatial dependencies and the Informer [46], a Transformer designed for long-range time-series forecasting, to model temporal dependencies. Unlike existing approaches that rely on static spatial connections, our DyGAT model introduces an attention-based dynamic adjacency matrix that evolves over time, reflecting changing patterns in $PM_{2.5}$ concentrations. In addition, we engineered directional edge features based on geolocation, wind direction and wind speed. These edge features highlight important features that affect pollutants dispersion across the region and provide directionality to the spatial relationships. The model will be trained and evaluated using the benchmark dataset 'Beijing Multi-Site Air-Quality Dataset' [54], which provides comprehensive air quality measurements and meteorological data from multiple locations in Beijing.

The second objective of this study is to investigate the effect of incorporating additional geospatial data about nearby pollution sources on forecasting accuracy of the hybrid spatiotemporal forecasting model. To address this, we will use an air quality dataset from the city of Nablus in Palestine, as provided by Saleh et al. [55]. This dataset includes details on pollution sources categorized by their hazard levels. We will first evaluate the model using only air quality-related features and then integrate the pollution source data to assess its impact on forecasting accuracy at each station. To accommodate the small size of the Nablus dataset, the temporal forecasting component of the hybrid model, represented by the Informer, was replaced with a Seq2Seq LSTM model.

The study aims to enhance spatiotemporal $PM_{2.5}$ forecasting by developing a model that effectively captures dynamic spatial dependencies and temporal patterns. It also examines how incorporating additional information about nearby pollution sources for each station influences forecasting accuracy.

1.4 Study Hypothesis

We hypothesize that integrating a dynamic variant of Graph Attention Network (DyGAT) to capture spatial dependencies with the Informer model for capturing the temporal dependencies will create a hybrid spatiotemporal forecasting model that improves $PM_{2.5}$

concentration forecasting. Specifically, we expect that this hybrid model will outperform models that use only temporal forecasting without spatial dependency integration.

In the case study using local data from Nablus, Palestine, we anticipate that incorporating additional contextual features about nearby pollution sources will enhance forecasting accuracy.

1.5 Importance of the Study

This study has a substantial importance for several reasons. It addresses a public health issue by enhancing the ability to predict PM2.5 concentrations accurately Additionally, it contributes to the field of environmental science and deep learning by exploring novel techniques for PM2.5 forecasting, which can be applied to various regions worldwide. Finally, it investigates how integrating geospatial and pollution sources information into spatiotemporal PM2.5 forecasting models affects forecasting accuracy, providing insights that have implications for air quality research and environmental policies.

Chapter Two
Foundational Concepts

This chapter presents the theoretical and mathematical foundations of the components that form the DyGAT-Informer model. The following sections provide a comprehensive overview of the key components, including Attention Mechanisms, Graph Attention Networks, and Time-series Transformers. These components are the base of the model's architecture. Each section includes relevant mathematical equations and theoretical concepts that are crucial for understanding the model's design and functionality.

2.1 Attention Mechanisms

During the construction of the DyGAT model, we used different attention mechanisms for different tasks within the model. The final version of the DyGAT model uses three different attention mechanisms; therefore, we will present an introduction to attention mechanisms before presenting the ones we used in the Methodology chapter.

In the context of deep learning, an attention mechanism refers to a computational mechanism that enables neural networks to selectively focus on certain parts of the input data while ignoring others, similar to the cognitive attention mechanism in the human brain that focuses on important elements in the environment while ignoring non-relevant information [56]. Attention mechanisms in neural networks help models weigh the importance of different input elements dynamically. They are general mechanisms so they are used in different types of deep learning architectures in computer vision, Natural Language Processing (NLP) and in other models that work with sequential data [57].

Although attention mechanisms have existed for a long time, their popularity started to rise after the year 2015, where they were used in many studies in machine translation and image captioning. In that time attention mechanisms were typically used with recurrent or convolutional layers in LSTMs and CNNs. However, in 2017 the Transformers [43] were introduced, they were built entirely using self-attention, which is a type of attention mechanisms [58].

Attention mechanisms have different types and variations of these types. One model can be built by combining different types of attention techniques. Several surveys [56], [57],

[58], presented different taxonomies for attention mechanisms types, we summarized them as follows:

- Soft and Hard Attentions: Soft attention assigns weights to each element in the sequence, allowing the model to focus on multiple parts simultaneously. While hard attention, chooses a single element to attend to at each step, making a more definitive choice. Soft attention is more commonly used in the literature.
- Self-Attention: Computes attention weights within the same sequence, allowing each element to attend to other elements, including itself. It is widely used in sequence modeling tasks like machine translation and sentiment analysis.
- Multi-Head Attention: Employs multiple sets of attention weights to capture different aspects or representations of the input. This enables the model to attend to various parts of the input simultaneously, providing richer context.
- Scaled Dot-Product Attention: A form of self-attention where attention scores are computed by taking the dot product of query and key vectors, followed by scaling and softmax normalization. Commonly used in Transformer-based models.
- Local Attention: Focuses on a subset of nearby inputs rather than the entire input sequence. This can improve efficiency, particularly for long sequences.
- Global Attention: Considers the entire input sequence when computing attention weights, potentially attending to all elements in the sequence. Useful for capturing long-range dependencies.

It should be noted that there are other ways to classify attention mechanisms. The taxonomy we presented is a summary of the four surveys we chose. These attention mechanisms are usually used in encoder-decoder architectures, like RNN and its variants LSTM and GRU, and in Transformers, specifically self-attention. They are also used in other architectures like CNNs, Memory Networks, GNNs and hybrid architectures. Each attention mechanism variation has its own customized mathematical representation. We will provide a generalized mathematical representation for attention mechanisms.

A general attention mechanism computes a context vector c_i for each input element i based on its relevance to the current context. To compute c_i we need a Query matrix (Q)

and a Key matrix (K), they could have different names depending on the model's architecture or attention mechanism type, but generally, they are defined as:

- Query: Represents the element of interest for which we want to compute the attention scores. It serves as a reference point or the focus of attention.
- Key: Keys are a representation of other elements in the input sequence. Each key is compared with the query to determine how relevant it is to the query. Keys help in assessing the importance or relevance of different elements in the input sequence with respect to the query.

To compute the context vector, we first need to compute attention scores also known as energy (e):

$$e_{ij} = f(q_i, k_j) \quad (2.1)$$

Where *f* is called a score function – also called compatibility or alignment function. The score function determines the similarity or compatibility between a query and a key, ultimately influencing the attention weights assigned to each key. There are different types of score functions used in attention mechanisms. Examples of score functions are: Additive, Multiplicative (dot product), Scaled multiplicative, Concat, Location-based, Similarity and Cosine-similarity based score functions [56] [59].

After calculating the attention scores, we then calculate the attention weights by applying softmax function to the attention scores,

$$\alpha_{ij} = \text{softmax}(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{j=1}^{n} \exp(e_{ij})} \quad (2.2)$$

Finally, we compute c_i as follows:

$$c_i = \sum_{j=1}^{n} \alpha_{ij} \, . \, v_j \quad (2.3)$$

Where v_j denotes the value or feature representation of input element j, and n is the total number of input elements.

2.2 Graph Attention Networks

A graph is a data structure used to represent relationships between objects. It is made up of nodes (or vertices) and edges. Nodes are the objects or entities, while edges show the relationships between them. Edges can be either directed, where the relationship has a direction from one node to another, or undirected, where the relationship is bidirectional between two nodes without a specific direction. Graphs can also be weighted or unweighted. In a weighted graph, edges have different weights, which can add another layer of meaning to the connections. Unweighted graphs, on the other hand, treat all edges equally [60].

An adjacency matrix is a way to represent a graph using a grid. It is a square matrix where each cell at position (i, j) indicates whether there is an edge between node i and node j. For a directed graph, the cell value shows the presence and direction of the edge, while for an undirected graph, it just shows the presence of an edge. In a weighted graph, the cell can also contain the weight of the edge. This matrix provides a compact and convenient way to store and work with graph data [60]. Graphs are versatile and can be used in algorithms like Dijkstra's for finding the shortest paths, and in machine learning models like Graph Neural Networks (GNNs) to handle complex relationships in data.

Graph Neural Networks [61] are a class of deep learning models specifically designed to work with graph-structured data, like transportation networks, social networks, and biological data (e.g. genes, proteins, etc.). Unlike traditional neural networks, GNNs are designed to excel at capturing the relationships and dependencies between connected nodes in a graph, making them particularly effective for tasks where the structure of the data is important.

Graph Attention Network (GAT) [38] was introduced in 2018 as a GNN variant that uses attention mechanisms for learning features on graphs. Some GNN variants like vanilla GCNs do not utilize attention mechanism, so they aggregate information from the node's neighbors equally, assuming all neighboring nodes have the same influence on one node. In contrast, GATs use attention mechanisms to focus on key features of neighboring nodes. Each neighboring node is assigned a different weight. GATs do not require a previous knowledge of the graph structure. The first GAT presented by Veličković et. al. [38] used self-attention for node classification tasks in graph-structured data. They also

used Multi-head attention with concatenation of the attention heads outputs to provide stability for the learning process.

GATs in general refer to a class of GNNs that utilize attention mechanisms to dynamically weight the importance of neighboring nodes and learn complex, context-dependent relationships within a graph. There are various types of GATs, with different taxonomies that categorize them based on factors like attention mechanisms, architecture, and application. Several surveys have been conducted to explore different types of GATs, providing detailed taxonomies and classifications. For instance, A survey [62] done to the attention mechanisms used in GNNs, classified the attention mechanism used in Veličković et. al. [38] work as "learnable attention" where the attention weights are learned. The survey also identified two other classifications of attention mechanisms in the literature. The first is "Similarity-based attention" which is also a learned attention but it allocates greater attention to objects sharing more similar hidden representations or features. The second classification is "Attention-guided walk", which is a type that utilizes attention mechanisms to guide the traversal process, unlike traditional random walks that traverse the graph uniformly or according to predetermined rules.

Another Survey [59] classifies attention mechanisms in GNNs into a two-level taxonomy. The upper level divides attention mechanisms into three types based on their high-level architectural differences: Graph Recurrent Attention Networks (GRANs), Graph Attention Networks (GATs), and Graph Transformers. GRANs focus on integrating RNNs with attention mechanisms for graph data. GATs, introduces attention directly into graph nodes, allowing nodes to weigh their neighbors' importance. Graph Transformers, leverages transformer-based architectures for graph data. The lower level of the taxonomy categorizes attention mechanisms in GNNs based on architectural designs within the three main categories.

Finally, a comprehensive review paper [63] categorized Graph Attention Networks (GATs) into six main types:

1. Global Attention Networks: which focus on the overall graph structure.
2. Multi-Layer GATs: which utilize multiple layers for deeper feature extraction.

3. Graph-embedding GATs: which utilize graph-embedding techniques to learn richer and more informative node representations.
4. Spatial GATs: which incorporate spatial information for more accurate modeling.
5. Variational GATs: which incorporate variational inference to effectively model complex, heterogeneous, and multimodal data across various domains.
6. Hybrid GATs, which combine various strategies for enhanced performance.

GATs have become widely utilized across various domains due to their ability to capture complex relationships within graph-structured data. They are particularly effective in node and graph classification, where they classify individual nodes or entire graphs based on their features. GATs excel in link prediction, where they estimate the likelihood of connections between nodes. Additionally, they are utilized in recommendation systems, where they enhance user-item interactions and predict user preferences. GATs are also used in traffic forecasting, where they model road networks to forecast traffic patterns. Moreover, they are used for molecular graph analysis, where they predict molecular properties. In image analysis, GATs improve tasks like segmentation and object detection by capturing spatial relationships. GATs are applied in medical fields for disease prediction and analysis of biological data, as well as in natural language processing, enhancing sentiment analysis by capturing contextual dependencies in text. Finally, GATs are employed in anomaly detection, including fraud detection and network security. These varied applications demonstrate the versatility and effectiveness of GATs in handling graph-based tasks across multiple fields [59][63].

Despite the effectiveness of GATs in handling graph-based data, they face several challenges. These challenges include computational complexity, as their cost increases with larger graphs, particularly due to the attention mechanism's complexity, leading to scalability issues. GATs can also suffer from over-smoothing in deep architectures, where node features become indistinguishable across layers, and they may struggle with capturing long-range dependencies in large graphs. Additionally, overfitting is a concern, particularly when data is limited or noisy. Moreover, interpretability remains a challenge, as understanding the reasoning behind attention weights can be difficult. Other limitations include high memory consumption due to attention weight storage and vulnerability to noisy data, which can reduce the robustness of GATs performance [59][62][63].

2.3 Time-series Transformers

The introduction of Transformers [43] has revolutionized various fields including natural language processing and image recognition. Self-attention mechanism, enabled them to capture long-range dependencies within sequences, and they have proven to be highly effective in modeling complex relationships [64].

The basic architecture comprises of the following components:

- Encoder: The encoder consists of multiple identical layers (blocks), where each one contains a multi-head self-attention mechanism and a position-wise feed-forward neural network.
- Decoder: The decoder also comprises of multiple identical layers, however, in addition to the self-attention and feed-forward network present in the encoder blocks, each decoder block incorporates cross-attention, which is used to attend to the encoder's output.
- Self-Attention Mechanism: The self-attention mechanism allows a token in the input sequence to attend to all other tokens in the same sequence. This mechanism enables the model to capture long-range dependencies efficiently.
- Positional Encoding: Transformers do not inherently understand the sequential order of tokens, which is why positional encoding is usually used to add positional information to the input embeddings. This allows the model to differentiate between the positions of tokens in the sequence.

The Transformers ability to work with long-range sequences made them a desirable candidate for building time-series forecasting models. In the past few years, many time-series Transformer variants have emerged. Time-series Transformer models presented different modifications to the vanilla Transformer to make it suitable for time-series forecasting. These modifications were done at different architectural levels. One survey [64] divided the modifications to the vanilla Transformer in time-series Transformer into two main categories, either modifications to the existing architectures, or new architectural innovations.

The first modification to the existing components is presenting new positional encoding methods like “Learnable Positional Encoding”, where the model can learn the positional

encoding from the input sequence. Another positional encoding technique is “Time-stamp Encoding”, where the time related features (hour, day, year, holidays etc.) are used as a positional encoding method.

The second type of modifications to the original components is modification to the attention module. The original Transformer has a memory and time complexity of $O(N^2)$, N is the length of the sequence, this poses a computational bottleneck when dealing with long-sequences. So some time-series Transformers added sparsity to the attention mechanism to reduce the memory and time complexity, examples of these Transformers are Informer [46] , LogTrans [65] and Pyraformer[66] and many others.

An example of architectural innovations to the Transformer is presented by the Informer, which has incorporated max-pooling layers between attention blocks. On the other hand, Pyraformer utilizes a C-ary tree-based attention mechanism.

Other time-series forecasting Transformers used signal decomposition to enhance the model’s forecasting abilities like Autoformer [67] and FEDfromer [68]. In addition, they have presented novel attention mechanisms, where the Autoformer introduced a novel auto-correlation mechanism that analyzes the data's periodicity to identify and aggregate similar sub-series, which enables the model to capture dependencies within the data more efficiently. FEDformer was built on the Autoformer, however it has introduced its own Frequency Enhanced Attention (FEA) mechanism.

Chapter Three
Methodology

In this chapter, we will present a thorough description of the datasets used in the study and provide insights acquired during the exploratory data analysis. In addition, we will explain the pre-processing techniques used to prepare the data for training and testing. Moreover, we will provide a detailed breakdown of the architecture of the proposed model.

3.1 Data Description and Preprocessing

In this study, the Beijing Multi-Site Air-Quality Dataset [54] will be the main dataset used to train and test the proposed model. The dataset contains a collection of spatiotemporal air quality measurements and meteorological data across 12 monitoring stations in Beijing. The Beijing air quality data was collected from March 1st, 2013, to February 28th, 2017.

The second dataset that will be used is an air quality dataset collected from 8 measuring stations in Nablus city in Palestine. The data was collected from January 6th 2022, to March 3rd 2022. It includes meteorological data for the city of Nablus, obtained from the 'Time and Date' weather website, covering the same period as the air quality dataset. This dataset was collected as a part of a study by Saleh et al. [55], where they presented a methodology for selecting air quality monitoring locations based on low-cost sensors and Geographic Information Systems (GIS) [69].

The distances between the measuring stations in the Nablus air quality dataset are considered small, where the longest distance between a pair of nodes is 10.93 km. Therefore, the weather data was the same for the whole city of Nablus, hence the only differences between nodes' features are the geolocation (longitude and latitude) and $PM_{2.5}$ readings. However, we chose this data to study another factor that is usually missing in other air quality datasets, which is the information about nearby pollution sources.

A general description of both datasets is presented in Table (1). One notable difference between the two datasets is the length of the time-step, where the Beijing dataset has a

time-step of 1 hour while the Nablus dataset has a time-step of one minute for $PM_{2.5}$ and 30 minutes for meteorological data.

Table 1

Datasets Properties

Properties	Beijing	Nablus
Number of stations	12	8
Number of Features	16	11
Number of Records	420768	$PM_{2.5}$: 542607; Meteorological Data: 2531
Timestamp Interval	1 Hour	$PM_{2.5}$: 1 Minute; Meteorological Data: 30 Minutes
Time Span	March 1st, 2013, to February 28th, 2017	January 6th 2022, to March 3rd 2022

Although the Nablus dataset contains a high number of records due to the $PM_{2.5}$ data being recorded every minute, the final number of records per station is 2692, and the total number of records across all stations is 21536. This reduction is due to the meteorological data being recorded at a 30-minute interval. For the Beijing dataset, there are 35064 records per station, and the total number of records is 420,768.

3.1.1 Data Analysis

The Beijing dataset consists of measurements from 12 stations, to perform data analysis, the data from all 12 stations were combined. The raw dataset contained 18 features, including 16 numerical features and two categorical features. Out of the 16 numerical features, four features represent temporal information (year, month, day, hour), one feature is the ordinal number of the timestamp. The remaining 11 numerical features are related to meteorological and air pollutant features. The two categorical features are "station name", "wind direction". Two additional features, longitude and latitude, were added for each station. Wind direction is a categorical feature with 16 categories, where for example, 'W' means west, 'SW' means southwest, and 'WSW' means west-southwest.

Table (2), presents a statistical summary of the relevant numerical features. The summary includes count, mean, minimum, maximum, standard deviation and Coefficient of Variation (CV) for each feature. The statistical summary provides valuable insights into

the distribution and range of these variables. For example, Atmospheric Pressure (PRES) demonstrates low variability in the data points with a CV of 1.04%. Features such as PM_{10}, NO_2, TEMP, and WSPM exhibit moderate variability, suggesting a noticeable but not extreme spread in their data values.

Table 2

Statistical Summary for Numerical Features in Beijing Dataset

Feature	Description	count	mean	min	max	std	CV
PM10	PM_{10} concentration in $\mu g/m^3$	415037	104.57	2.00	999.00	91.70	87.67%
SO2	Sulfur dioxide concentration in $\mu g/m^3$	412682	15.82	0.29	500.00	21.63	136.74%
NO2	Nitrogen dioxide concentration in $\mu g/m^3$	409675	50.64	1.03	290.00	35.08	69.25%
CO	Carbon monoxide concentration in $\mu g/m^3$	401843	1229.30	100.00	10000	1157.82	94.21%
O3	Ozone concentration in $\mu g/m^3$	409210	57.31	0.21	1071.00	56.55	98.67%
TEMP	Temperature in degrees Celsius	420390	13.54	-19.90	41.60	11.44	84.50%
PRES	Atmospheric Pressure in hPa	420395	1010.75	982.40	1042.80	10.47	1.04%
DEWP	Dew point temperature in degrees Celsius	420385	2.49	-43.40	29.10	13.79	553.01%
RAIN	Rain precipitation in mm	420398	0.06	0.00	72.50	0.82	1366.67%
WSPM	Wind speed in m/s	420464	1.73	0.00	13.20	1.25	72.25%
PM2.5	$PM_{2.5}$ concentration in $\mu g/m^3$	412954	79.74	2.00	999.00	80.74	101.25%

High variability is observed in SO_2, CO, O_3, and $PM_{2.5}$ with high CVs, indicating a broad range of data points. Notably, DEWP and RAIN show extremely high variability, with CVs that suggest substantial fluctuations relative to their means.

To better understand the temporal variations in $PM_{2.5}$ concentrations, Figure (1), presents a bar chart depicting the average $PM_{2.5}$ concentration across different months from 2013 to 2017. Each bar represents the average $PM_{2.5}$ concentration for a particular month. It seems that there are high variabilities in the $PM_{2.5}$ average values for the same month over the years. A clear example is February, where in 2014 it had the highest average $PM_{2.5}$ concentration of the year, while in 2016 it had the lowest average concentration of the year.

Figure 1

Average $PM_{2.5}$ Concentration by Month for Each Year for Beijing Dataset

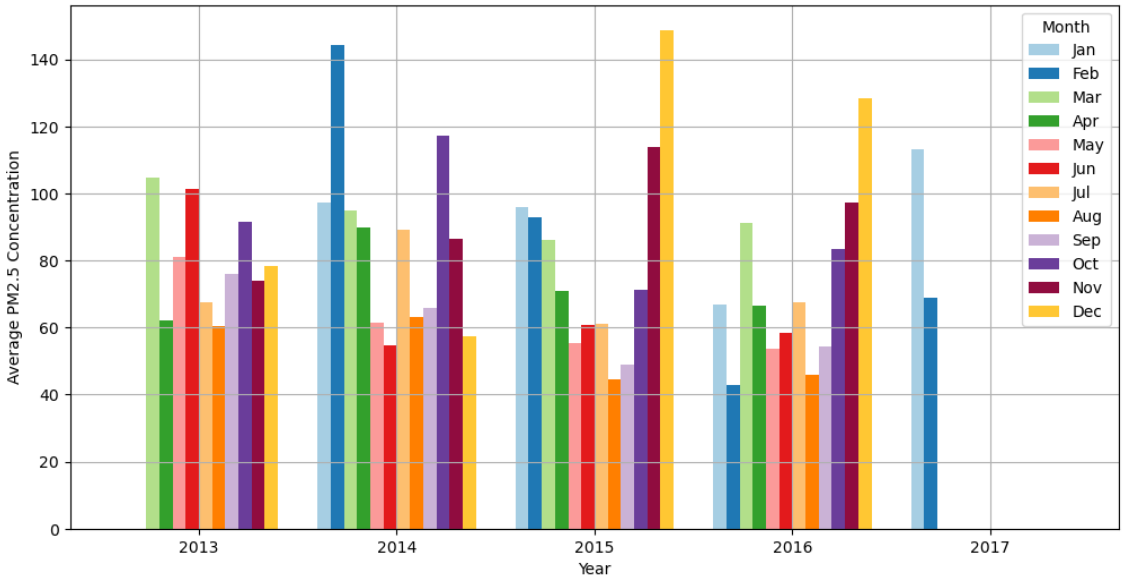
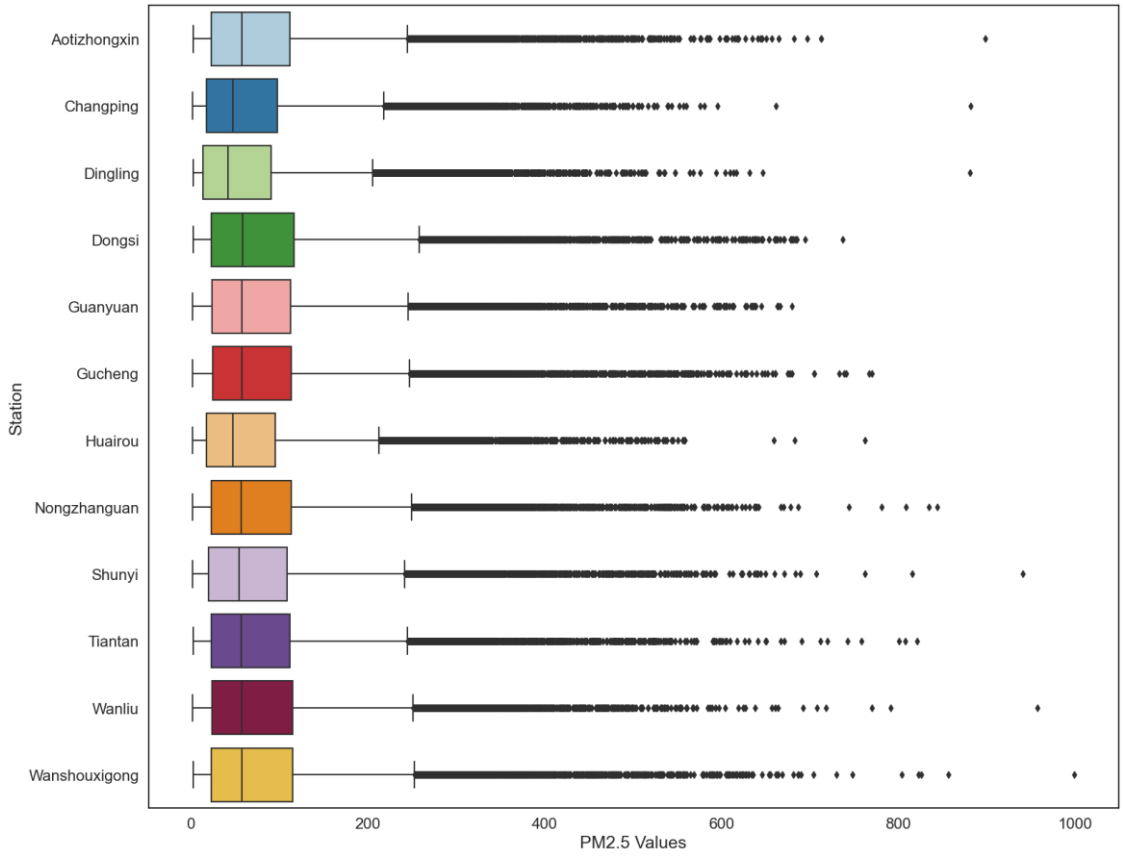


Figure (2) shows boxplots for the $PM_{2.5}$ values at each station for the Beijing dataset. The boxplots visualize the distribution of the data. They offer insights into our data, including the 25th percentile (first quartile (Q1)), the median (Q2), the 75th percentile (third quartile (Q3)). They also highlight the minimum and maximum values, as well as the presence of outliers. Each horizontal boxplot represents a specific station, and the vertical axis displays $PM_{2.5}$ values. The box for each station provides a summary of the data distribution. The elements of the boxplot are as follows [70]:

- Box: It represents the Interquartile Range (IQR) which extends from the first quartile (Q1) to the third quartile (Q3), representing the middle 50% of the data. The line within the box indicates the median (Q2) $PM_{2.5}$ value.
- Whiskers: The lines extending from the box indicate the range of the data within 1.5 times the IQR.
- Outliers: Individual data points plotted beyond the whiskers are considered outliers.

Figure 2

The Boxplots of the $PM_{2.5}$ Values at Each Station in the Beijing Dataset.



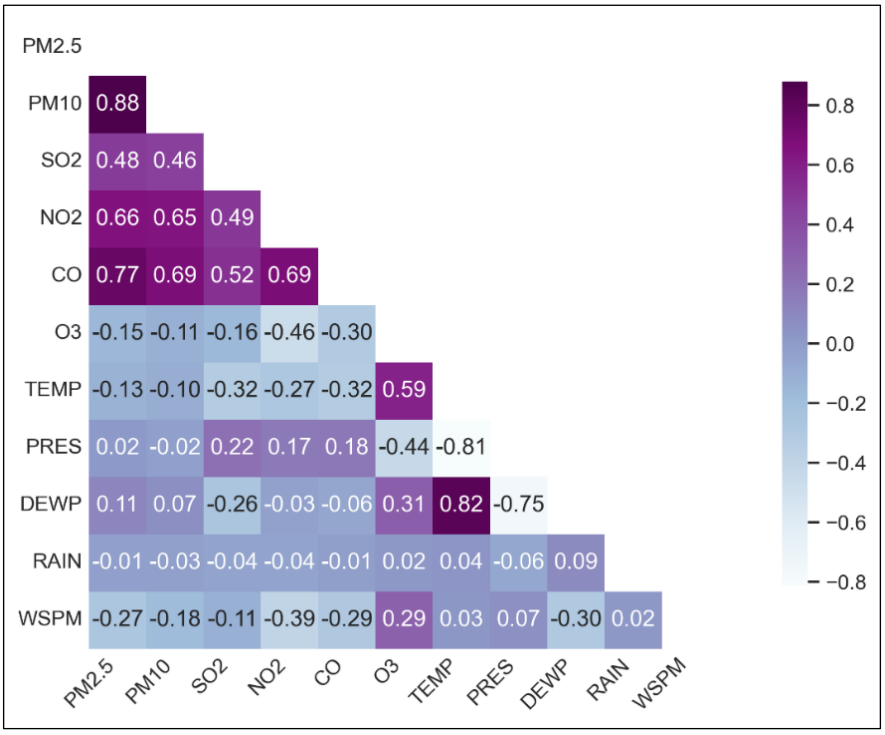
Most stations show similar distributions in their $PM_{2.5}$ values, with many having their median values close to 50. There are many outliers especially at higher $PM_{2.5}$ values, which indicates some spikes in pollution levels higher than the typical range for each station. Some stations have wider IQRs and more outliers, indicating greater variability in $PM_{2.5}$ values. The Wanshouxigong station has the highest outlier value (close to 1000), suggesting an episode of severe pollution.

We have decided to keep the outliers in the data because in the context of environmental data like $PM_{2.5}$ levels, outliers often represent real and extreme events like pollution level spikes due to weather conditions, industrial activities, or traffic. In this case, including outliers allows our model to learn and adapt to these extreme events, providing more realistic and comprehensive understanding of air quality dynamics. This will lead to a more robust model that can provide better forecasts. Secondly, air quality forecasting models are often used in scenarios where predicting extreme events is crucial. Which makes including outliers in the training data essential, especially for cases where the forecasting model is used in early warning systems.

To ensure that these extreme values are not measurement errors or noise, we examined these extreme $PM_{2.5}$ level across all stations. We observed that these extreme values occurred consistently across all measuring stations, making the likelihood of an instrument error very low. Additionally, the extreme values remain within the possible range for $PM_{2.5}$ concentration levels. Figure (3) shows the correlation matrix for the Beijing dataset, which provides a visual representation of the relationships between the numerical features in the Beijing dataset.

Figure 3

The Correlation Matrix of the Numerical Features in the Beijing Dataset.



The matrix highlights both positive and negative correlation coefficients (r), with darker colors indicating stronger positive relationships. $PM_{2.5}$ shows a strong positive correlation with PM_{10} (r = 0.88), CO (r = 0.77), and SO2 (r = 0.66), which suggests that these pollutants' levels are related. One possible reason for the high positive correlation between these pollutants is having similar sources such as vehicle emissions and industrial activities.

We have decided to include all of the features when training the hybrid spatiotemporal $PM_{2.5}$ forecasting model to fully leverage the comprehensive nature of the dataset. Each feature, such as various air pollutants ($PM_{2.5}$, PM10, NO2, CO, O3, SO2), meteorological data (temperature, pressure, dew point, wind speed, wind direction, precipitation), and temporal information, offers unique insights into the complex atmospheric processes affecting $PM_{2.5}$ levels. The DyGAT-Informer hybrid model is well-equipped to handle feature interactions, even those involving highly correlated variables. GAT's attention mechanism allows it to focus on the most relevant relationships between features, while Informer's self-attention mechanism identifies and prioritizes the temporal dependencies that matter most. By utilizing the full feature set, the model can capture both direct and indirect interactions among features, thereby enhancing its ability to forecast $PM_{2.5}$ levels with greater accuracy.

Regarding the Nablus dataset, as stated before the dataset was created by combing the $PM_{2.5}$ readings from 8 locations in Nablus city and weather data retrieved from the weather website. The dataset from Nablus contains 11 features, including 8 numerical features, 2 categorical features, and 1 temporal feature. The numerical features are latitude and longitude (spatial coordinates), temperature, humidity, atmospheric pressure, visibility distance, rain and PM2. The categorical features are station name and 'Weather'. And finally, the temporal feature represents the timestamp of each measurement.

The categorical feature 'Weather' is a description of the weather in phrases separated by dot (e.g. "Light rain. Partly cloudy."). The 'Rain' feature is binary, with a value of zero indicating no rain and one indicating rain. The textual description of the amount of rain is stated in the 'Weather' feature.

Table (3) presents statistical summary for the numerical features. There seems to be some errors in the measurements, because the maximum value in $PM_{2.5}$ (17188.39 µg/m^3) is not reasonable, and the minimum value is (-1) which is also not possible.

Table 3

Statistical Summary for Numerical Features in Nablus Dataset

Feature	Description	count	mean	std	min	max	CV
Temp	Temperature in Fahrenheit.	21536	54.81	6.291	37	81	11.48%
Wind speed	Wind speed in mph.	21536	7.44	4.90	0	37	65.8%
Humidity	Humidity as a percentage.	21536	0.71	0.16	0.18	1	23.1%
Barometer	Atmospheric pressure in "Hg.	21536	30.04	0.12	29.74	30.39	0.42%
Rain	Binary (raining or not).	21536	0.41	1.14	0	1	278.4%
$PM_{2.5}$	$PM_{2.5}$ concentration in µg/m^3	21536	22.85	137.64	-1	17188.39	601.6%

The CV analysis reveals that 'Barometer' feature has low variability in data points, while Temp, Wind speed, and Humidity show moderate variability. Finally, $PM_{2.5}$ have high variability, indicating substantial fluctuations.

For the Nablus dataset we had to deal with some extreme outliers that do not represent real pollution level but rather a measurement error. For example, at the "Unit_F_Hijjawi" station, the maximum $PM_{2.5}$ value was 17188.39 µg/m^3, which is highly improbable. In addition, the minimum value at the "118_NNUH" station was -1, which is not possible because $PM_{2.5}$ values cannot have negative values.

3.1.2 Data Preprocessing

The first step in preprocessing the data is examining the number of missing values for each feature, and filling them with an imputation method. For the Beijing dataset, the

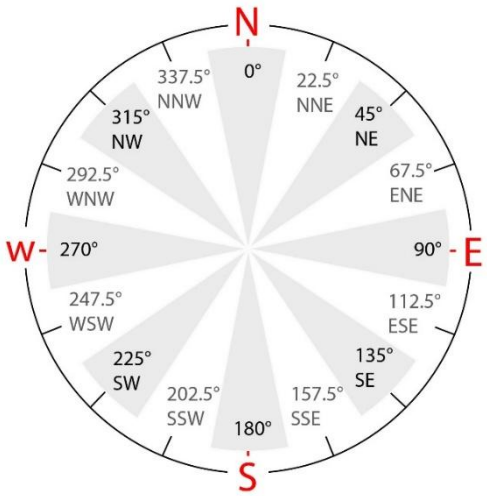
missing values for meteorological features were less than 0.1%. However, for pollutant features, the missing values were mostly around 2%, except for CO, which had 4.92%, and O3, which had 3.16%. For the Nablus dataset, the only two features with missing values were wind speed, with 0.04%, and visibility, which had a much higher percentage of missing values at 44.76%.

Since this is a time-series data, simple imputation methods like mean and median may not be suitable because a value at one time-step is related to values around it more than values at distant time-steps. For the numerical features, an Iterative Imputation method was used, which uses a multivariate imputation algorithm to estimate missing values iteratively. It treats each feature's column with missing values as a target variable and uses the other columns as predictors to estimate the missing values. At each iteration, the Iterative Imputer uses a regression model to predict the missing values. For the categorical features like wind direction, a k-nearest neighbors Imputer was utilized, which is used for imputing missing values in datasets using the k-nearest neighbors algorithm. It imputes missing values based on the values of neighboring data points, and it can handle categorical data.

In the Beijing dataset, we encoded the wind direction by converting the textual description of the direction into angular degrees, so we ended up with 16 different degrees to describe the wind direction. Figure (4) shows the wind direction mapping from textual categories to degrees.

Figure 4

Wind Direction Degree Mapping Chart.



The $PM_{2.5}$ measurement in the Nablus dataset had temporal granularity of 1 minute, while the meteorological measurements had a 30 minutes temporal granularity. We aggregated $PM_{2.5}$ measurement into 30 minutes time interval to match the meteorological data.

As stated in the section [Data Analysis], the Nablus dataset had some outliers due to measurement error, so we had to fix them. The negative $PM_{2.5}$ value of (-1) was changed to zero. Moreover, the extremely high value was treated as missing and filled during the imputation process. Regarding the 'Visibility' feature (visibility distance), it was removed due to approximately 45% of the values being missing, and the majority of the non-missing entries having a value of 10 miles, offering little variability.

We applied min-max normalization to the data, to ensure a uniform scaling of features and improve model performance, especially when the features have different ranges. Min-max normalization transforms each feature to a common scale by mapping its values to a range between 0 and 1 using this formula:

$$x_{norm} = \frac{x - \min(x)}{\max(x) - \min(x)} \quad (3.1)$$

where min(x) and max(x) are the minimum and maximum values of the feature, respectively. Other normalization methods were explored during the initial experiments, and min-max normalization yielded the best results in terms of evaluation metrics.

3.1.3 Features Engineering

In the previous section, we addressed the data pre-processing techniques that have been used like filling the missing values, normalization and encoding of categorical data. In this section, we will present the feature engineering techniques we used in both datasets. Feature engineering involves creating new features or modifying existing ones to improve the performance of machine learning models.

Starting with the Beijing dataset, we engineered edge features for each pair of nodes to be used in DyGAT, alongside the node features. This approach provides an additional perspective on the dynamic structure of the graph. It incorporates various edge features, including directional ones that influence the dispersion of $PM_{2.5}$. These edge features encapsulate the spatiotemporal relationships between monitoring sites by leveraging the geographical coordinates of the nodes and the wind features at both the source and target

nodes. First, we calculated the geographical distance between each pair of monitoring stations utilizing the Haversine formula to take the curvature of the Earth's surface into consideration for more precise distance calculations. Then, we added wind speeds at both the source and target nodes, also we included the difference in wind direction between source and target node.

Here is the Haversine formula:

$$a = \sin^2\left(\frac{\Delta_\phi}{2}\right) + \cos e(\phi_1) \cdot \cos(\phi_2) \cdot \sin^2\left(\frac{\Delta_\lambda}{2}\right) \quad (3.2)$$

$$c = 2 \cdot \operatorname{atan2}(\sqrt{a}, \sqrt{1-a}) \quad (3.3)$$

$$d = R \cdot c \quad (3.4)$$

where:

- ϕ_1 and ϕ_2 are the latitudes of the two points in radians.
- $\Delta_\emptyset$ is the difference in latitudes (ϕ_1- ϕ_2).
- Δ_λ is the difference in longitudes ($\lambda_1 - \lambda_2$).
- R is the Earth's radius (mean radius = 6,371 km).
- d is the distance between the two points.

We wanted to capture the influence of wind direction on pollutant dispersion. To do this, we used the geo-locations of each pair of nodes to calculate the initial bearing. Then, we compared the wind direction to determine if the wind flow from the source node was directed towards the target node. If the wind direction aligns with the calculated bearing within a defined threshold of 45 degrees, it suggests that the wind is blowing towards the target location. The following equations are used to determine if the wind from source node blows toward target node:

$$y = \sin(\Delta_\lambda) \cdot \cos(\phi_2) \quad (3.5)$$

$$x = \cos(\phi_1) \cdot \sin(\phi_2) + \sin(\phi_1) \cdot \cos(\phi_2) \cdot \cos \Delta_\lambda \quad (3.6)$$

$$\theta = \operatorname{atan2}(y, x) \quad (3.7)$$

$$\text{Intitial Bearing} = \left(\left(\theta \times \frac{180^\circ}{\pi}\right) + 360^\circ\right) \bmod 360^\circ \quad (3.8)$$

$$\Delta_{\text{wind}} = |\text{Wind Direction (source)} - \text{Initial Bearing}| \quad (3.9)$$

Then adjusting Δ_{wind} for the circular nature of wind directions:

$$\Delta_{wind} = \begin{cases} \Delta_{wind} & if\ \Delta_{wind} \leq 180^\circ \\ 360^\circ - \Delta_{wind} & if\ \Delta_{wind} > 180^\circ \end{cases} \quad (3.10)$$

$$\text{Wind Blows Towards Target} = \begin{cases} 1 & if\ \Delta_{wind} \leq 45^\circ \\ 0 & if\ \Delta_{wind} > 45^\circ \end{cases} \quad (3.11)$$

Where:

- $\phi_1$and ϕ_2 are the latitudes of the two points in radians.
- $\Delta_\emptyset$ is the difference in latitudes (ϕ_1- ϕ_2).
- Δ_λ is the difference in longitudes ($\lambda_1 - \lambda_2$).
- θ: The initial bearing angle in radians.

In summary, we have five features for each edge, and they are:

1- Distance: The Haversine distance between the two nodes.

2- Wind direction difference: The difference in wind directions, measured in degrees, between the two nodes. This value provides additional context about the dynamic spatial relationship between a pair of nodes. It is a scalar value, representing the angular difference between the wind directions at the two locations.

3- Wind blows towards target: It is a binary feature, where 1 means the wind form source node is directed towards the target node and 0 means it is not.

4- Wind speed at source node.

5- Wind speed at target node.

As for the Nablus dataset, the weather features were the same for all stations because they are very close to each other and located within a relatively small city, so the distance between each pair of nodes was the only edge feature, which is a static feature unlike the temporal nature of the edge features in the Beijing dataset.

After processing the Beijing dataset, we turned our attention to the Nablus dataset, which contains a different set of node features. One of the features in the Nablus dataset is 'Rain,' which is a binary value indicating whether it is raining or not. In addition, there is

another feature called ‘Weather’, which contains a textual description of the weather like the sky condition (e.g. clear, cloudy, scattered clouds, etc.) and rain intensity. Therefore, we extracted the rain intensity description from the ‘Weather’ feature and added them to the ‘Rain’ feature. Both features were then encoded using an ordinal encoder. The ‘Rain’ feature had 6 ordinal categories ranging from ‘No Rain' to 'Heavy rain’. Now, the ‘Weather’ feature contains categorical description of the sky, which was converted to an ordinal description with 7 categories representing cloud coverage.

We made two versions of the Nablus dataset, one version with $PM_{2.5}$ values and meteorological features, and another one with added features from Saleh et al. [55] study. The study used three main categories of criteria to create the PM hazard map for Nablus’ potential sources of air pollution, which are factories, quarries and traffic. Additionally, other factors influencing PM distribution like altitude, wind speed and wind direction, were also considered. Then, they calculated the distance and direction in degrees, from each pollution source to the measurement stations. The distance and direction of pollution hazard sources were added to the second version of the data, to test the influence of the added context about nearby pollution sources on the model’s performance.

3.2 Model Architecture

In this section, we present our hybrid spatiotemporal forecasting model, the DyGAT-Informer, which combines two components: the novel Dynamic GAT (DyGAT) model that we have designed to capture dynamic spatial dependencies between measuring stations at different time steps, and the Informer Transformer, which captures the temporal dependencies at each station. The informer [71] was chosen due to its ability to model complex and long-range temporal information.

Before describing our primary hybrid model, it should be noted that for the case study dataset, the 'Nablus Dataset' we combined DyGAT with a Seq2Seq LSTM to create DyGAT-LSTM. This model initially served as one of the baseline models to compare the performance of DyGAT-Informer, trained with the Beijing dataset, with models that used other types of temporal components. This choice was taken to accommodate the small size of the Nablus dataset, which was insufficient to train the Informer. In the case study, DyGAT-Informer was included among the baseline models. We made this adjustment because the main goal of using the case study data, despite its small size, was to test the

second hypothesis of the study: that including contextual information about nearby pollution sources for each station would improve forecasting accuracy. The DyGAT-LSTM model is described in detail in section [Baseline Models] in chapter 4 [Experimental Design and Setup].

When we started building our primary hybrid model, many candidate time-series Transformers were considered including Informer, Autofromer and FEDformer. Each of these Transformers is a powerful forecasting model, but each has its own strengths and weaknesses depending on the task and the nature of the dataset. During the preliminary experimentation and hyper-parameter tuning phase, we found that the Informer was the most suitable model to our dataset. We included the Autoformer as a part of the final baseline models to compare the two Transformers' performances. We decided to exclude the FEDformer model after the preliminary experiments, as its results were similar to those of the Autoformer. However, it had a significantly higher computation time, with each epoch averaging 9 minutes and 30 seconds, compared to just 13 seconds for the Informer and 16 seconds for the Autoformer.

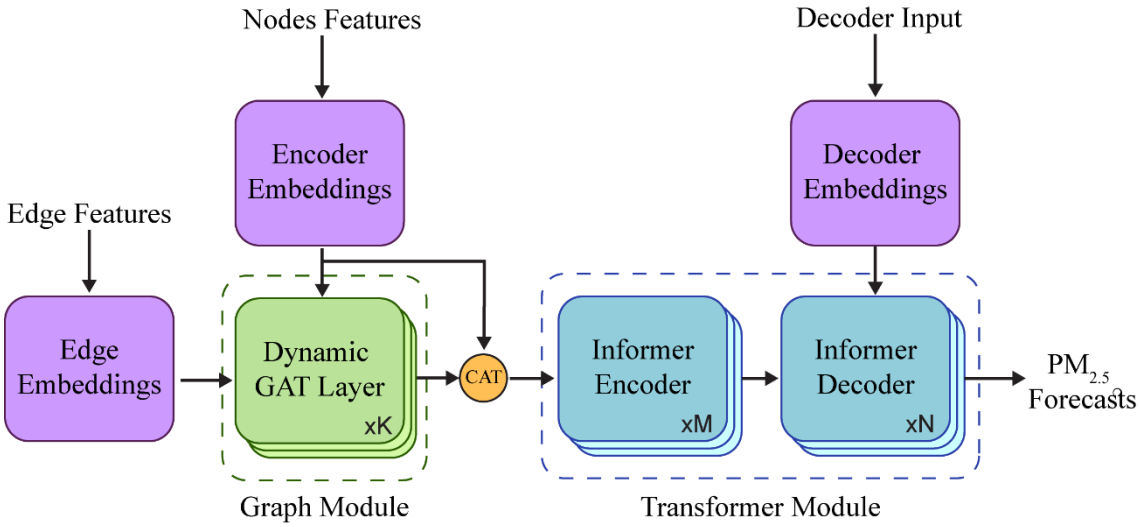
When it comes to spatial dependencies modeling, we hypothesized that the relations between each pair of nodes is not static, and not only determined by the geo-location of the nodes, but rather a dynamic relation that changes with the nonlinear processes of the air dynamics. To model the dynamic spatial relations between the nodes, we chose GAT because we can include different attention mechanism to assign varying weights to the nodes and edges in the graph.

As mentioned in the feature engineering section, we created edge features to help the GAT model in capturing the dynamic spatial relation between each pair of nodes. The edges have a direction, meaning that edge features between a pair of nodes is different depending on which node is the source and which one is the target. The difference in edge features based on direction is because that one of the features determines whether the wind from the source node is directed towards the target node or not. The edge features also contain information about the wind speed and difference in wind direction between nodes, which are time-varying features. Wind direction and speed affect the $PM_{2.5}$ dispersion in the atmosphere that is why we chose them as edge features to emphasize their role in forecasting future $PM_{2.5}$ concentrations.

Figure (5) presents a general overview of the DyGAT-Informer model. The 'Graph Module' represents our DyGAT model and the 'Transformer Module' represents the Informer. There are multiple ways to configure a hybrid model. Each configuration has its own set of advantages and disadvantages, depending on the specific goals and requirements of the forecasting task. We experimented with both sequential and parallel configurations to integrate the DyGAT and Informer components of the hybrid model. We found that the optimal configuration involved sequential fusion, where the DyGAT processes the data first, and its output is then fed into the Informer. Additionally, we examined whether to feed the DyGAT output directly into the Informer or concatenate it with the original nodes embeddings, and we found that using concatenation produced better results. In the figure, the concatenation process is depicted by the 'CAT' block.

Figure 5

Overall Architecture of the DyGAT-Informer Model.



There are two historical sequences used as input to the first stage of the model, and they are the nodes and edges features. Assuming that p is the number of past time-steps considered in the historical input sequence, Xn and Xe are vectors representing the nodes features and the edge features respectively then the input sequence for the nodes features is Xn=$\{Xn_{t-p+1}, Xn_{t-p+2}, \ldots, Xn_t\}$, and the input sequence for the edge features is Xe=$\{Xe_{t-p+1}, Xe_{t-p+2}, \ldots, Xe_t\}$. Each one of these inputs is combined with temporal features of that sequence (minute, hour, day, week day and year), and used to create the initial node and edge embeddings that utilize temporal positional encoding to help the

Informer in understanding the tokens positions in the sequence. In figure (5), Xn is the 'Encoder Input' and node embeddings are 'Encoder Embeddings'.

Regarding the 'Decoder Input', during training, the input Yn is the future (target) sequence the model needs to forecast, where Yn = $\{Xn_{t+1},\ Xn_{t+2}, \ldots, Xn_{t+H}\}$, with H representing the forecasting horizon This input is combined with temporal features and used to generate the 'Decoder Embedding'. For inference, the decoder relies on previously predicted values, while still incorporating temporal embeddings to preserve temporal context. The decoder employs attention mechanisms that includes masking, to ensure it attends to relevant parts of the sequence and prevents future information from being accessed during training. In addition to 'Decoder Embeddings', the Informer decoder also receives encoded representations from the Informer encoder as another input.

In our model, we converted the input time-series features into embeddings that contains two parts:

1. Token Embeddings: In this type of embeddings, the raw input data is converted into a dense, continuous vector representation. Token embeddings are used to represent individual data points in a high-dimensional space. This dense vector is a form of representation that the Transformer architecture can effectively process.
2. Temporal Embeddings: Which represent temporal features (hour, day, month, and year) that can help the model understand periodic patterns and time-based dependencies. They are added to the token embeddings.

The Informer Transformer code [72] provides three types of temporal embeddings as a hyper-parameter**:** **'**timeF'**,** **'**fixed'**,** and 'learned'. They are used to embed temporal features of the data (like hour, day, week, month, etc.) into a high-dimensional space. Here's what each type represents:

1. Fixed Embedding**:** It creates non-trainable embeddings based on trigonometric functions (sine and cosine). These embeddings are dependent on the position of time steps and are generated using a predefined formula without any learnable parameters. This method captures the periodicity of time, which helps in modeling temporal patterns in the data.

2. Learned Embedding: The temporal information is embedded into a vector that can be learned during training. In this method, the temporal features (e.g., hour, day, month) are passed through an embedding layer, and the model learns the best way to represent these features during the training process.
3. Time Feature (timeF) Embedding: It takes raw time-related features and applies a linear transformation to embed these features into a higher-dimensional space. Instead of using predefined or learned embeddings, the raw time features (like "hour", "day of the week", etc.) are directly encoded by feeding them into a linear layer. This approach does not learn a specific embedding for each temporal position but linearly transforms the temporal features into a vector.

Our model uses 'fixed' temporal embeddings, as they yielded better results during hyper-parameter tuning.

3.2.1 DyGAT

We reviewed different GAT models that have been used in spatiotemporal forecasting tasks, and we found a spatiotemporal traffic-forecasting model called STGAT [73]. Their implementation of the “Graph Attention Layer” in the model was the same as in the original GAT [38]. Their “Graph Attention Layer” have self-attention mechanism and a learnable adjacency matrix, which they called “self-adaptive adjacency matrix”. It is a parameterized module with learnable parameters. An element in the adjacency matrix adj(i, j) specifies the weight between node i and node j. During training, the adjacency matrix is adjusted iteratively to adapt to the data.

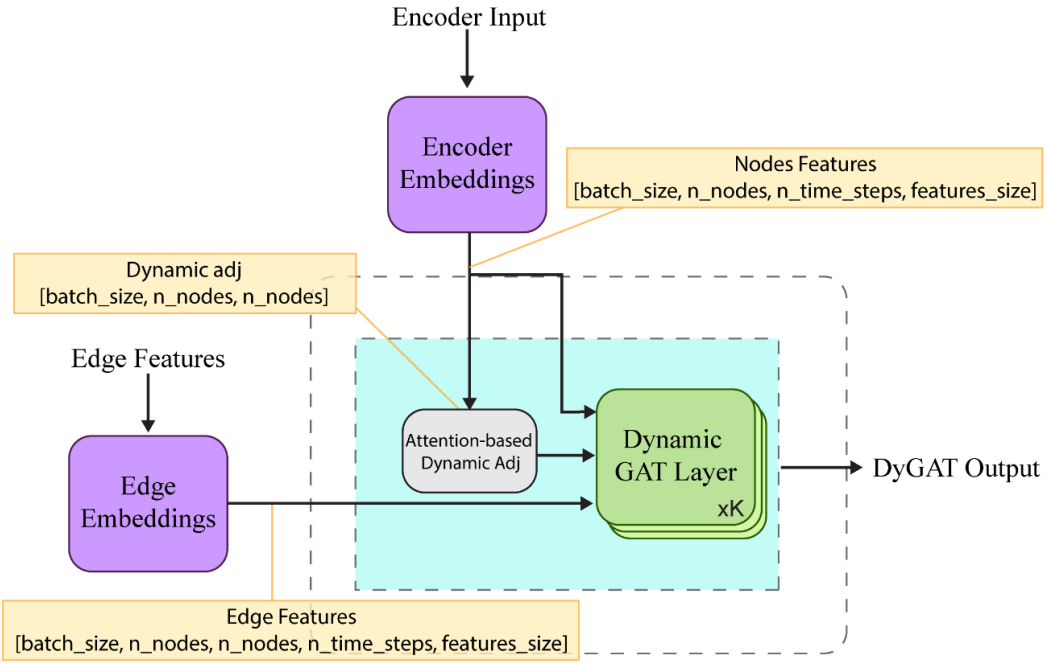
We used their Graph Attention Layer as a base to build our DyGAT model on top of it. As for the adjacency matrix, we created our own “Attention-based Dynamic Adjacency Matrix”, which uses attention mechanisms to create a weighted adjacency matrix for the graph. We kept the original self-adaptive adjacency matrix as hyper-parameter. During the hyper-parameter tuning process, the model showed better performance when our attention-based adjacency matrix was used.

The DyGAT model consists of a stack of dynamic GAT layers and an attention-based dynamic adjacency matrix. Each DyGAT layer have self-attention mechanism called ‘node attention’ that captures the dynamic spatial relations between nodes based on their

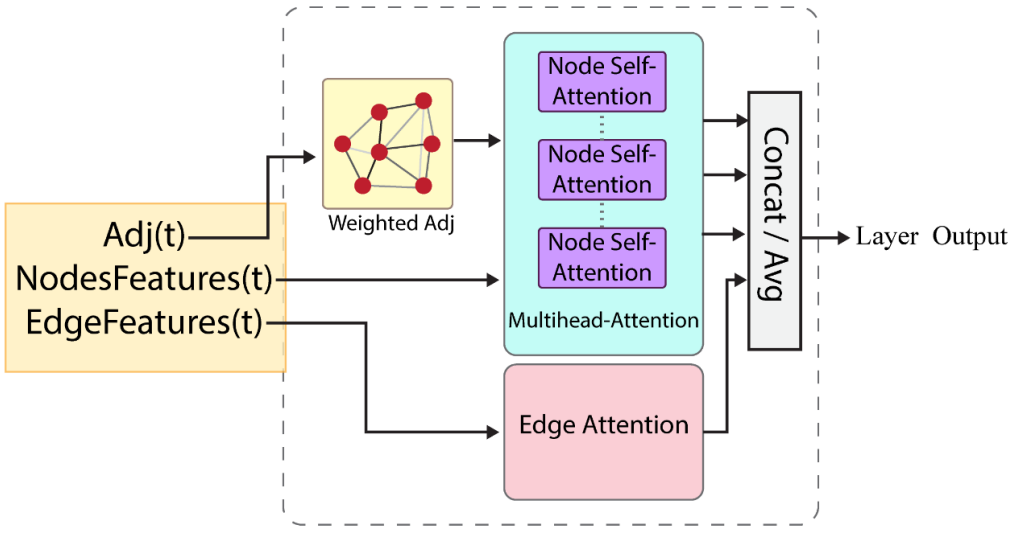
own features using the attention-based dynamic adjacency matrix. Multi-head attention was utilized in node attention computations. The layer also contains an 'edge attention' weights computed based on the dynamic edge features, these attention weights are used to enhance the spatial decencies representations between the nodes. Figure (6.a) shows a general view of the DyGAT model, and Figure (6.b), shows the structure of the dynamic GAT layer.

Figure 6

DyGAT Architecture. (a) Overall Architecture of the DyGAT. (b) Detailed Architecture of a Dynamic GAT Layer.



(a) DynamicGAT Module



(b) DynamicGAT Layer

There are two inputs to the DyGAT model: ‘Encoder Embeddings’ and ‘Edge Embeddings’. The ‘Encoder Embeddings’ tensor is of shape [batch_size, n_nodes, n_time_steps, feature_size], where:

- batch_size refers to the number of samples processed in a single training or inference step.
- n_nodes corresponds to the number of nodes in the graph.
- n_time_steps represents the number of time steps in the input sequence.
- feature_size represents the dimensionality of the feature vector, capturing both node attributes and temporal features transformed into an embedding space.

The ‘Edge Embeddings’ tensor is of shape [batch_size, n_nodes, n_nodes, n_time_steps, feature_size], where the two n_nodes dimensions indicate source and target nodes, encoding their relationships over time.

At each time step, the Encoder Embeddings are extracted and represented as a tensor of shape [batch_size, n_nodes, feature_size]. This tensor is first used as input to the Dynamic Attention-based Adjacency Matrix, which computes a weighted adjacency matrix for that specific time step. The resulting weighted adjacency matrix, along with the Encoder Embeddings for that time step, serves as input to the first DyGAT layer. Similarly, the Edge Embeddings at the same time step are extracted as a tensor of shape [batch_size, n_nodes, n_nodes, feature_size] and is also used as input to the first DyGAT layer. The following sub-sections provide a detailed description for the DyGAT components.

3.2.1.1 Attention-based Dynamic Adjacency Matrix

The purpose of using attention to compute the adjacency matrix is to dynamically adjust the graph structure based on the input features at each time-step. This dynamic adjustment allows the model to capture the changing relationships between nodes in the graph over time. The dynamic adjacency matrix applies linear transformations to the node features in order to obtain query and key representations then computes attention scores using dot product attention mechanism, and applies softmax to obtain a normalized adjacency matrix.

The DyGAT uses one time-step at time to compute the dynamic adjacency matrix. The input node features Xn at time-step t is of shape [batch_size, n_nodes, nodes_feature_size]. First, Query (Q) and Key (K) are computed as follows:

$$Q = X \,.W_Q \quad (3.12)$$

$$K = X \,.W_K \quad (3.13)$$

Where, W_Q and W_K are learnable weight matrices. Then we compute the attention weights (A) using softmax function,

$$A = \text{softmax}\left(\frac{Q.K}{\sqrt{d_k}}\right) \quad (3.14)$$

Where d_k is the dimension of the key vectors. Now we construct the adjacency matrix:

$$Adj = A \times A^T \quad (3.15)$$

3.2.1.2 Dynamic GAT Layer

This is the core component in the DyGAT module, where two sets of attention mechanisms are used to capture the spatial dependencies between nodes at each time-step. The first attention mechanism 'nodes attention' is a graph self-attention mechanism similar to the one used in the original GAT, and in STGAT [73], the difference here is that STGAT used adjacency matrix that is initialized as a learnable parameter, while we utilized attention mechanism to create a true dynamic adjacency matrix that changes with each time-step.

After the dynamic adjacency matrix is created, node attention uses it as the basis for computing attention scores between nodes. The adjacency matrix influences which nodes are considered neighbors and how much weight each neighbor gets when aggregating features. To compute node attention weights, the Q and K are first computed using equations (3.12) and (3.13), then the attention scores are computed using leaky ReLU function,

$$e_{ij} = \text{LeakyReLU}(Q_i . K_j) \quad (3.16)$$

where e_{ij} represents the attention score between node i and node j, and leaky ReLU is define as follows:

$$\text{leakyReLU}(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \quad (3.17)$$

Then, the attention scores are used to compute the weighted sum of neighboring node representations:

$$\text{nodes\_attention} = \sum_j \text{softmax}(e_{ij}).X_j \quad (3.18)$$

This process is repeated across different attention heads then the outputs of the multi-head attention are either concatenated or averaged, depending on the user's choice, so we made this a hyper-parameter.

The second attention mechanism 'edge attention', which is applied to the edge features we created during the feature engineering phase. We reviewed different attention mechanism and found that using additive attention [74] was the most suitable due to the nature of the directional edge features. These directional features are specific to each pair of nodes and require a more focused attention mechanism that can address these pairwise dependencies.

The edge features input vector Xe at time-step t is of shape [batch_size, n_nodes, n_nodes, edge_feature_size]. First, Q and K are computed using equations (3.12) and (3.13), and then the Q and K are combined using an additive operation and passed through a hyperbolic tangent activation function (tanh):

$$Z = \tanh(Q + K) \quad (3.19)$$

After that, Z is passed through another linear transformation to obtain attention scores, this transformation is done through a simple neural network called 'Linear' in PyTorch and it is defined like this:

$$Y = xA^T + b \quad (3.20)$$

Where x is the input tensor, A is a learnable weight matrix, and b is bias. After passing Z to the linear layer, we get the attention scores, which are then passed through a softmax function to get the edge attention weights.

The edge attention weights are then used to modulate the nodes features to get another perspective of the spatial dependencies this time based on the edge features. Then the output is added to the multi-head attention outputs to be concatenated or averaged. By combining node self-attention to model how each node influences all other nodes, and edge additive attention to capture the weight of interactions between each pair of nodes based on directional features, DyGAT can create a richer spatiotemporal representation.

3.2.1.3 Final DyGAT Output

Each Dynamic GAT Layer returns a modified representation of the input that includes the spatial dependency for multiple time-steps. The final vector represents a spatiotemporal representation of the entire input sequence, which will be either the final output of DyGAT or the input to the next Dynamic GAT Layer, if multiple layers are used.

3.2.2 Informer

The Informer [46] is a time-series Transformer variant introduced by Zhou et. al. to solve some of the issues in the Transformers when they are used for long-range time-series forecasting. The Informer presented three major modifications to the vanilla Transformer. The first modification was introducing ProbSparse self-attention mechanism to reduce the Transformer's quadratic time complexity. It has achieved a O(Llog L) time complexity. ProbSparse self-attention reduces the time complexity by selectively attending to the most relevant parts of the data.

The second major modification was creating a self-attention distilling technique to efficiently handle extremely long sequences. Finally, the model employs a generative style decoder that predicts long time-series sequences in a single forward operation to enhance the speed of inference for long-sequence predictions significantly.

The Informer model is composed of an encoder and a decoder, both of which utilize a combination of multi-head self-attention and ProbSparse self-attention layers. The encoder processes the input sequence, while the decoder generates the predicted output sequence. Here is a more detailed description of the Informer architecture:

1. Encoder: It processes a sequence of input features to effectively capture the temporal dependencies. It employs multiple layers of multi-head ProbSparse self-attention. It utilizes self-attention distilling to addresses redundancy in the encoder's feature maps, by prioritizing features with dominant information. It reduces the time complexity significantly through a max-pooling operation and convolutional layers. The number of self-attention distilling layers decreases progressively in each layer, forming a pyramid structure.

2. Decoder: It generates long sequential outputs efficiently. It uses the standard Transformer decoder with layers of multi-head self-attention. It utilizes “Masked Multi-head Self-Attention” to prevent the decoder from attending to future tokens during training to maintain autoregressive property. The decoder incorporates generative inference, where it generates the entire output sequence in a single forward pass to accelerate the decoding process.

The output of our DyGAT model is concatenated with the original input embeddings to provide stability during training, and provide the informer with a version of the original temporal data before adding the dynamic spatial dependencies representations. The user can choose to directly use the DyGAT output as input to the Informer, or to concatenate it with the input embeddings. We made the concatenation between DyGAT output and the input embeddings as an option in the hyper-parameters, however, as mentioned before, during hyper-parameters tuning we found that concatenation produced a better performance, so it is the default option. The overall architecture of the Informer is presented in Figure (A.1) in Appendix A.

Chapter Four
Experimental Design and Setup

This chapter outlines the experimental design and setup employed to evaluate the performance of the proposed hybrid spatiotemporal $PM_{2.5}$ forecasting model. We begin by introducing the baseline models against which our model's performance will be evaluated. We will provide an overview of each model's architecture and the rationale for choosing them. Following this, the experimental setup are described, including hardware and software specifications, model training and testing parameters, hyper-parameters tuning process and the evaluation metrics that were used.

4.1 Baseline Models

The first phase of the study aimed to test the first hypothesis, which stated that using a hybrid model addressing both spatial and temporal dependencies in air quality data would result in more accurate $PM_{2.5}$ forecasts at individual stations compared to using only a time-series forecasting model. To evaluate this, we employed our primary model, DyGAT-Informer. We used the Beijing dataset to train both our DyGAT-Informer model and an Informer model without DyGAT to verify whether capturing spatial dependencies between the measuring stations would improve forecasting results.

In addition, we also combined our DyGAT model with other time-series forecasting models to evaluate the performance of the temporal component of the hybrid model. These models are a Seq2Seq LSTM model and Autoformer. In addition, we compared our model with STGAT [73] because we used their version of GAT as a foundation of our model. We believed that this comparison would provide insights into the effectiveness of the modifications in our DyGAT variant. Finally, we chose a PyTorch implementation [75] of a model [76] that uses GCN as their spatial dependency computation module.

The following is a detailed description of the architectures for the four baseline models:

1- DyGAT-Autoformer

This hybrid model combines our DyGAT with the Autoformer [67], a time-series forecasting Transformer that utilizes signal decomposition technique that breaks down the time-series data into component representing trend, seasonal, and residual parts of the

time-series. Signal decomposition is supposed to help the model in understanding the underlying structure of the data. We selected this Transformer due to its high performance demonstrated in the literature. The model uses Autocorrelation Mechanism instead of self-attention mechanism used in Transformers. It focuses on identifying periodic patterns within the data, and their relationships, which enables the model to capture long-range dependencies.

2- DyGAT-LSTM

We combined our DyGAT model with an LSTM-based encoder-decoder model, which is typically used for sequence-to-sequence tasks like time-series forecasting. The Seq2Seq LSTM model was a part of a research paper about spatiotemporal wind speed forecasting model by Bentsen et. al [77]. Since LSTMs are popular and powerful time-series forecasting models, we found it interesting to combine an LSTM with DyGAT in a hybrid model and compare its performance to that of the DyGAT-Informer.

In this LSTM model, the encoder processes the input sequence using a network of multilayer LSTM, which transforms the input data into a series of internal representations, known as "hidden states" that capture the sequential dependencies and patterns in the input. These internal representations are then passed to the decoder. The decoder also uses multilayer LSTM network that generates the output sequence step-by-step by leveraging the information from the hidden states provided by the encoder. The model uses a training strategy called "recursive strategy", where each prediction is used as the input for the next time-step, allowing the model to iteratively refine its forecasts based on previous outputs.

3- ST-GAT

The ST-GAT model architecture combines a GAT with temporal convolution to handle spatiotemporal data. The model have a "TimeBlock" used to capture temporal dependencies for each node separately. The temporal convolution is done using Gated Temporal Convolutional Layer (GTCN), which helps the model learn temporal patterns by applying a series of convolutional operations over time. The TimeBlock have several GTCN layers. After the temporal processing the model uses GAT layers to capture spatial dependencies. The GAT layers use node attention mechanism to dynamically weigh the importance of neighboring nodes. After the temporal and spatial processing of the data, a

final output layer transforms the final feature representations into the desired forecast window. The overall ST-GAT architecture is presented in Figure (A.2 - a) in Appendix A, and the Figure (A.2 - b) illustrates the structure of GTCN.

4- ST-GCN

The Spatiotemporal graph convolutional network (ST-GCN), have similar design to ST-GAT, where the model have spatiotemporal blocks, and it uses temporal convolution to process the temporal dependencies. However, this model uses Graph Convolutional networks (GCN) instead of Graph Attention Networks (GAT). The architecture comprises two key components, the "TimeBlock" and the "STGCNBlock". The TimeBlock is used to handle the temporal dependencies of the data. It applies temporal convolutions to the input data. Temporal convolutions operate along the time dimension of the data, processing each node in the graph independently. The block uses three convolutional layers, where the outputs from these layers are combined through a series of non-linear transformations, including addition and activation functions, to capture the temporal dependencies.

The STGCNBlock combines temporal and spatial convolution. First, it applies a temporal convolution using the TimeBlock, then applies a spatial graph convolution, and finally applies another temporal convolution. The spatial convolution step is done by using a learnable weight matrix, where the model combines information about how the nodes in the graph are connected (using the adjacency matrix) with the temporally processed features. The overall model has two STGCNBlocks followed by a final TimeBlock. Finally, the model uses a fully connected PyTorch 'Linear' layer to map the output to the desired number of future time-steps. The ST-GCN architecture is presented in Figure (A.3) in Appendix A

4.2 Experimental Setup

We trained all of the models using the Beijing dataset on Kaggle platform using an NVIDIA Tesla P100 GPU. The variables environment was pinned to the date April 19, 2024, to ensure the same versions of the libraries are used in the training and testing process for all of the models. As for the case study, the models were trained using the Nablus dataset on a local machine (a laptop) not Kaggle. This laptop is equipped with an Intel Core i7-5500U CPU @ 2.40GHz processor and 12 GB of RAM. It runs Anaconda

as the Python distribution framework, with version 'conda 23.7.4', and uses PyTorch version '2.2.1+cpu'. The local machine was also used in the early stages of the preliminary experiments that used Beijing Dataset.

4.2.1 Model Training and Optimization

Model training and optimization are fundamental steps in the development of machine learning models. Training involves configuring the model with specific parameters and iteratively adjusting its weights based on the training data to minimize errors, and guiding the training process using training parameters. Optimization, on the other hand, focuses on fine-tuning the model's hyper-parameters to enhance its performance and ensure generalization to unseen data. The training parameters are shown in Table (4), we used early stopping mechanism, so if the validation loss did not improve for 5 epochs the training will terminate.

Table 4

Models Training Parameters

Parameter	Value
Batch size	$16^{(1)}/8^{(2)}$
# Train epochs	50
Early stopping patience	5
learning rate	0.001
Loss Function	$MAE^{(1)}/MSE^{(2)}$
Optimizer	ADAM

Note. (1) Parameter for Beijing dataset, (2) Parameter for Nablus dataset.

The learning rate was initially set to 0.001 and it undergoes a periodic decay by a factor of 0.8 during the course of training. The periodic decay ensures stable and efficient optimization, preventing the model from overshooting the optimal solution as it approaches convergence.

Each one of the forecasting models was trained and tested across three different forecasting windows. The time-step in the Beijing dataset is 1 hour, so the forecasting windows were 24 time-steps, 48 time-steps, and 72 time-steps, which are equivalent to 24, 48, and 72 hours, respectively. However, for the Nablus dataset, the time-step is 30 minutes, so the forecasting windows were 6 time-steps, 12 time-steps, and 24 time-steps,

which are equivalent to 3, 6, and 12 hours respectively. The choice of forecasting windows was determined through experimentation during the hyper-parameter tuning phase. The results of the forecasting windows during the tuning phase were influenced by the size of the datasets and their different set features.

The training and testing process for each model under each forecasting window was repeated three times and the average values of the evaluation metrics for each model were taken as the final evaluation metrics values. Repeating the training and testing multiple times is done to account for the variability of the results due to the stochastic nature of the training process and parameters initialization. We used the same data splits and set a fixed random seed to ensure consistency across different experiments.

The total training time for each model across different forecasting windows and datasets, does not accurately reflect the actual computation time for the model. Because the time taken for each of the three experiments for a given model and forecasting window was slightly different. This variation was primarily due to the use of the early stopping mechanism, which resulted in each experiment having a different number of training epochs. Also due to the stochastic nature of the models, the epoch time of each experiment slightly varied. To provide a more consistent comparison, we focus on average epoch times. By examining epoch times, we obtain a clearer picture of the model's computational efficiency.

On average, regardless of the forecasting window and dataset used, the epoch times for the five models can be ranked as follows: ST-GCN was the fastest, followed by ST-GAT, DyGAT-LSTM, DyGAT-Informer, and DyGAT-Autoformer. The epoch times for DyGAT-LSTM, DyGAT-Informer, and DyGAT-Autoformer were relatively close to each other and higher than those for ST-GCN and ST-GAT, suggesting that the temporal convolution used in ST-GCN and ST-GAT may have lower computation time compared to the other time-series forecasting models. Finally, the experiment time did not necessarily increase with longer forecasting windows for the same model. Instead, the early stopping mechanism sometimes led to cases where longer forecasting windows having less training time because the model converged earlier.

Table (5) presents the final hyper-parameters for each model, including those used when training the models with the Beijing dataset and Nablus datasets.

Table 5

Models Hyper-parameters after Tuning

Parameter	DyGAT-Informer	DyGAT-Autoformer	DyGAT-LSTM	ST-GAT	ST-GCN	Notes
embed_size	$64^{(1)}/32^{(2)}$	$64^{(1)}/32^{(2)}$	$64^{(1)}/32^{(2)}$	-	-	GAT Embed size.
d_model	$128^{(1)}/64^{(2)}$	$128^{(1)}/64^{(2)}$	$128^{(1)}/64^{(2)}$	-	-	Encoder/Decoder Embed size
trans_n_heads	$8^{(1)}/2^{(2)}$	$8^{(1)}/2^{(2)}$	-	-	-	# of transformers attention heads.
e_layers	$3^{(1)}/2^{(2)}$	$3^{(1)}/1^{(2)}$	2	-	-	# of encoder blocks.
d_layers	2	2	2	-	-	# of decoder blocks.
d_ff	512	512	512	512	64	Neural networks hidden size.
dropout	0	0.05	0.1	0.5	0.5	Temporal component dropout rate.
activation	gelu	gelu	relu	relu	relu	Activation function.
adj_type	attention	attention	attention	-	-	Type of adjacency matrix.
gat_hid_dim	$256^{(1)}/128^{(2)}$	$256^{(1)}/128^{(2)}$	$256^{(1)}/128^{(2)}$	64 - 512	16	Hidden dimension in GNN.
gat_out_dim	$64^{(1)}/32^{(2)}$	$64^{(1)}/32^{(2)}$	$64^{(1)}/32^{(2)}$	64	64	Graphs output size.
gat_n_heads	4	4	$4^{(1)}/2^{(2)}$	4	-	# of attention heads in GATs.
gat_n_layers	1	1	1	4	2	Number of graph layers.
edge_out	32	32	32	-	-	Size of edge embeddings.
graph_dropout	$0.05^{(1)}/0.1^{(2)}$	$0.05^{(1)}/0.1^{(2)}$	$0.05^{(1)}/0.1^{(2)}$	0.5	-	Graph dropout rate.
concat_attn_heads	TRUE	TRUE	TRUE	-	-	Concat or average attention heads in DyGAT.
st_concat	TRUE	TRUE	TRUE	-	-	Concate DyGAT output with input embeddings or not.

Note. (1) Hyper-parameter for Beijing dataset, (2) Hyper-parameter for Nablus dataset.

For hyper-parameter tuning, we first used Optuna [78], which is hyper-parameter tuning framework that employs various algorithms like Grid Search, Evolutionary Algorithm and Bayesian Optimization to efficiently sample the best possible hyper-parameter configurations from the defined search space. Through extensive experimentation with Optuna, we identified several promising hyper-parameter combinations. Then, we refined these combinations through meticulous manual tuning and rigorous experiments to pinpoint the optimal set of hyper-parameters. In addition to hyper-parameter tuning, we conducted a series of preliminary tests to determine the optimal configurations for the hybrid model. These tests examined whether to use data embeddings as input to the DyGAT or just the processed data, and evaluated whether placing the DyGAT before the Informer or between the Informer's Encoder and Decoder yielded better results. The final configuration of the hybrid model was presented and explained in chapter 3 'Methodology'.

4.2.2 Model Evaluation and Validation

In time-series forecasting, the temporal order of observations is crucial. To construct the training, validation, and test datasets, we employed a holdout method to ensure that the temporal dependencies inherent in our dataset are preserved. This method involves splitting the dataset in a way that reflects the sequential nature of time-series data, thereby avoiding potential information leakage that could occur if data points from different time steps were mixed.

Other methods, such as cross-validation, were considered for evaluating the model's performance. However, given the non-stationary nature of the air quality data, the cross-validation approach and its variants that were developed for time-series data present significant challenges. For non-stationary time series data like air quality measurements, maintaining the chronological order of observations is crucial to accurately reflect the model's forecasting capabilities. Therefore, a holdout method, which preserves this temporal sequence by training on earlier data and testing on subsequent data, is more appropriate [79].

The holdout method approach ensures that the model's performance is evaluated based on its ability to forecast future observations without any contamination from future data, thereby providing a realistic assessment of its forecasting accuracy. The substantial size

and temporal span of our dataset (420,768 records over four years) ensures that both the training and testing sets represent the seasonality and trends in air quality data. We used 80% of the data as a training set. The remaining 20% was further divided equally to create the validation and test sets.

When evaluating the performance of machine learning models, the choice of evaluations metrics depends on which aspects of the model's performance should be evaluated. Usually, multiple evaluation metrics are used to create a comprehensive evaluation of the model's performance. For this purpose, Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) and Symmetric Mean Absolute Percent Error (SMAPE) were chosen as the evaluation metrics [80] [81]. Both MAE and RMSE are scale-dependent metrics, meaning that their values depend on the range of the data. For this reason, both the actual and predicted values were scaled back to their original values before evaluation. In contrast, SMAPE is scale-independent and provides a normalized measure of error, making it suitable for comparing models across different datasets.

MAE is a robust and interpretable evaluation metric that measures the average magnitude of errors in the predictions, without considering their direction. It provides a clear indication of how far, on average, the model's predictions deviate from the actual values. MAE is particularly useful when the goal is to understand the model's average performance, as it treats all errors equally, making it less sensitive to outliers. The following is the MAE formula:

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \acute{y}_i| \quad (4.1)$$

Where y_i is the actual value, $\acute{y}_i$ is the predicted value and n is the number of test entries. RMSE is an evaluation metric that emphasizes larger errors due to the squaring of individual differences before averaging them. In the context of spatiotemporal $PM_{2.5}$ forecasting, where predicting extreme values accurately can be crucial, RMSE provides a more sensitive measure of the model's performance in capturing these rapid variations. The following is the RMSE formula:

$$\mathrm{RMSE} = \sqrt{\sum_{i=1}^{n}\frac{(\acute{y}_i - y_i)^2}{n}} \quad (4.2)$$

Finally, SMAPE is an evaluation metric that measures the accuracy of forecasts as a percentage of the actual values, providing a normalized error measurement. Unlike MAE and RMSE, which are in the same scale as the data, SMAPE offers a scale-independent measure that allows for comparison across different data scales. SMAPE is defined as:

$$\text{SMAPE} = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - \acute{y}_i|}{|y_i| + |\acute{y}_i|} \times 100\% \qquad (4.3)$$

By using MAE, RMSE, and SMAPE, the evaluation process captures a comprehensive view of the model's performance. MAE provides insight into the average magnitude of errors, RMSE highlights the significance of larger errors, and SMAPE offers a scale-independent assessment of forecast accuracy. Together, these metrics allow for a more nuanced understanding of how well the model performs in both typical and extreme conditions.

Chapter Five
Results and Discussion

In this chapter, we present and analyze the results of our experiments. First, we evaluate the performance of the primary DyGAT-Informer model in forecasting $PM_{2.5}$ levels across different forecasting windows using the Beijing dataset, assessing how it supports the first study hypothesis regarding spatiotemporal forecasting compared to temporal-only forecasting. We also compare the DyGAT-Informer with baseline models. Following this, we analyze the DyGAT-LSTM model trained on the Nablus dataset, both with and without the additional GIS data, to determine if the results support the second study hypothesis.. Additionally, we provide a comparison of all five models trained on the Nablus dataset.

5.1 Beijing Dataset

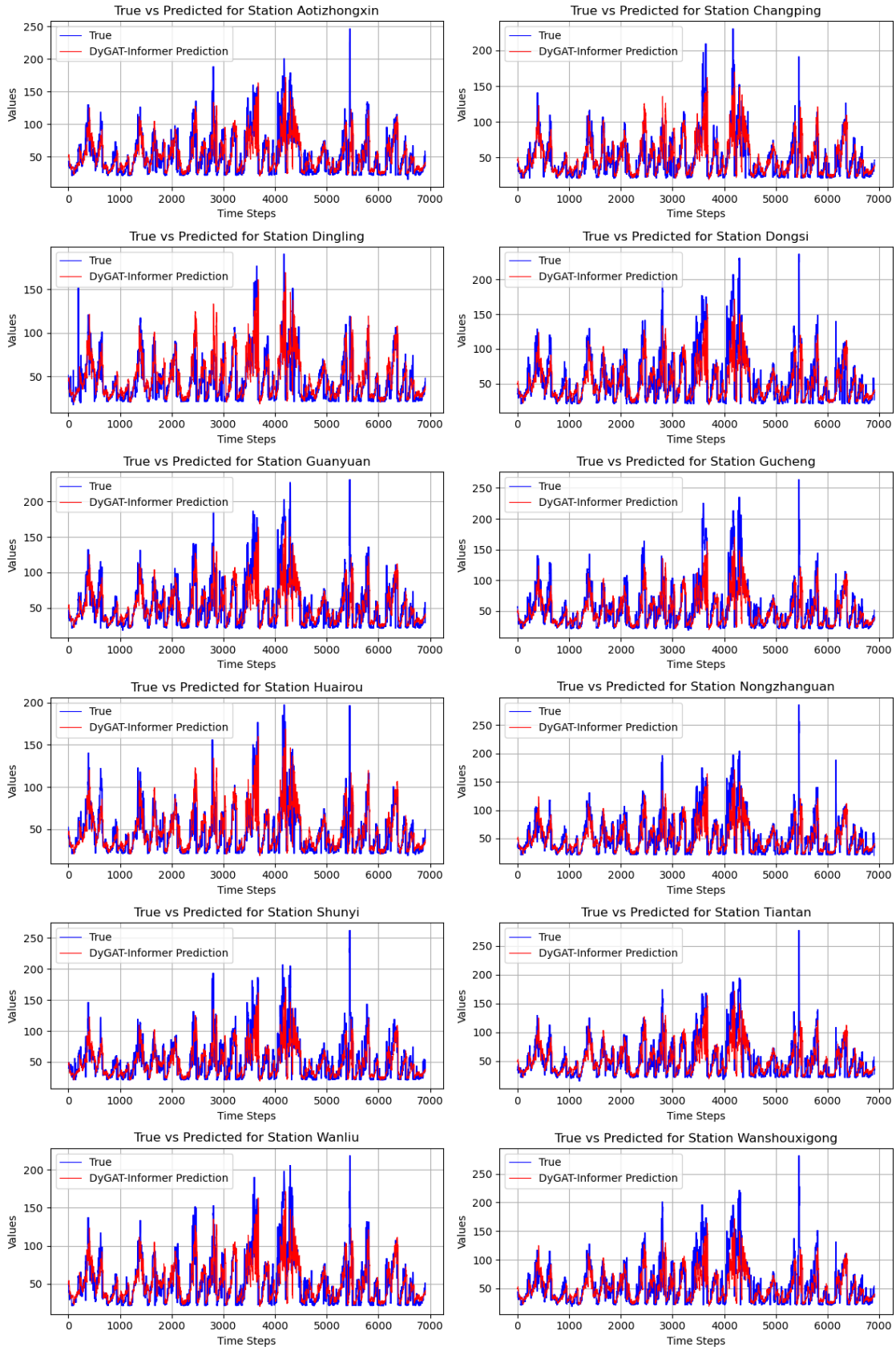
First, we used the Beijing dataset to train our DyGAT-Informer model. Figure (7), shows the actual and predicted values for a 24-hour forecasting window for each station. The x-axis represents the $PM_{2.5}$ concentrations, and the y-axis represents the time-steps. The actual values are presented in the blue line and the predicted values are in the red line.

The results in Figure (7) shows that the model was able to predict the future values with high precision and capture the general trend of $PM_{2.5}$ fluctuations across the stations. Although the model was generally able to create forecasts close to the actual values, still there were some discrepancies when $PM_{2.5}$ levels were high especially during periods of rapid change. The model underestimated the very high $PM_{2.5}$ values for most stations except for the Changping, Dingling and Huairou stations, where the model overestimated the $PM_{2.5}$ values in certain periods.

However, despite some discrepancies in the forecasts due to variability in the air quality data, the model was able to capture many of the patterns observed in the actual data. These discrepancies could be attributed to factors such as sudden changes in traffic patterns and industrial activities. Generally, the results suggest that the model is effectively learning and predicting the underlying temporal structure in the data.

Figure 7

Actual vs Predicted Values by DyGAT-Informer Model (Beijing Dataset)



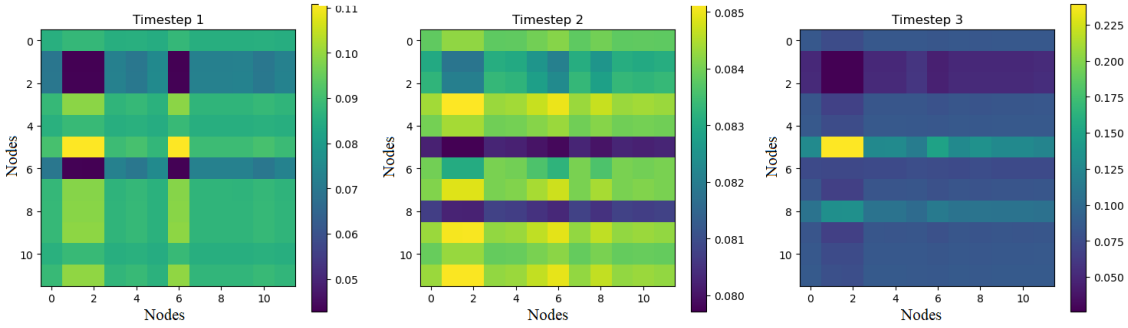
5.1.1 DyGAT Performance Analysis

To acquire insights about the DyGAT's ability to capture the dynamic spatial relations between the stations, we visualized both nodes attention weights and edges attention weights for the DyGAT used in the DyGAT-Informer hybrid model for the 24-hour forecast window. Figure (8) shows the heatmaps of both attention weights over three consecutive time-steps. The color intensity represents the attention weight, where warmer colors indicating higher attention weight.

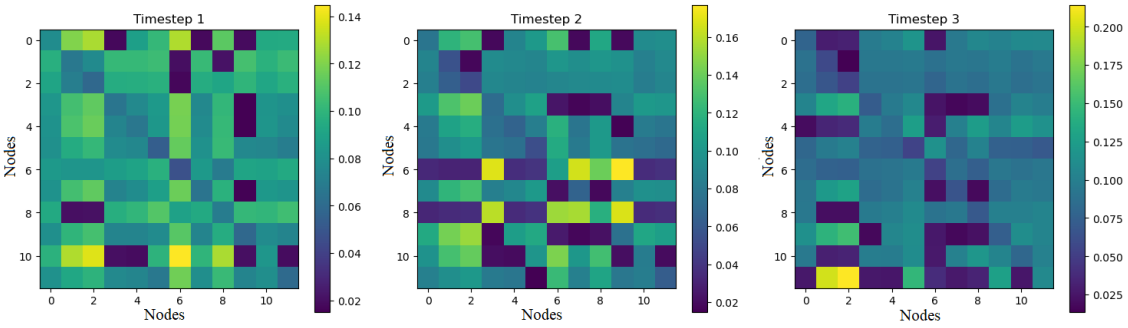
Figure 8

DyGAT Attention Weights Visualization. (a) Nodes Attention Weights at 3 Different Time-steps. (b) Edge Attention Weights at 3 Different Time-steps. (Beijing Dataset)

(a) Nodes Attention Weights



(b) Edges Attention Weights



The node attention heatmaps in Figure (8.a) shows heatmaps of the attention weights assigned by the DyGAT model to the graph nodes. These attention weights highlight the importance of certain monitoring stations at different time steps, reflecting their influence on the spatiotemporal $PM_{2.5}$ forecasting task. Nodes with consistently higher attention weights may correspond to locations where meteorological or pollutant conditions affect other locations significantly. In contrast, during periods of uniform node attention weights, it suggests a lack of strong spatial variability in meteorological and pollutant

data across the region. This could indicate relatively stable conditions across all stations during those time steps. The attention weights change across time-steps, indicating that the model is capturing the dynamic relationships between monitoring stations. This is crucial for our $PM_{2.5}$ spatiotemporal forecasting task as dependencies between locations can evolve over time. We can also observe that some nodes have similar spatial patterns especially the second station (Changping) and the third one (Dingling). In fact, these two stations are indeed the closest two, where the physical distance between them is 4.41 km.

Figure (8.b) shows heatmaps of the edge attention weights learned by the DyGAT model at three different time-steps. The attention weights vary across time-steps, reflecting the model's ability to capture evolving relationships between monitoring stations based on the edge features. It also indicates that the importance of certain connections varies depending on the wind speed and direction, which are the temporal edge features. These temporal variations of the edge attention weights demonstrate that the edge features we engineered had a significant role in capturing the spatiotemporal dependencies between the stations.

A higher edge attention weight between a pair of nodes could indicate a significant pollutant transport influenced by wind direction and speed at both nodes, especially when the wind direction from source node aligns with the calculated bearing between nodes as explained in equations (3.5 – 3.11). In contrast, low edge attention weights might indicate weak or negligible interaction between two stations due to misaligned wind directions or large geographic separation. The edge attention weights showed some spatial patterns similar to the ones observed in the nodes attention weights. For example, Changping and Dingling stations also had similar spatial patterns because of their physical proximity.

Nodes and edges attention weights from a different batch of data are presented in Figure (A.4). In Figure (A.4 - a), the node attention weights from this batch exhibit similar trends to those seen in Figure (8.a) for some time steps. However, in other time steps, the attention weights were less variable, with each node having a uniform effect on all other nodes. This could indicate that, during these time steps, the air quality and meteorological features across the region were relatively uniform, resulting in fewer strong spatial relationships to emphasize. The edge attention weights in Figure (A.4 - b) also display patterns similar to those in Figure (8.b). However, the homogeneous attention weights

observed, particularly in time steps 5 and 6 in Figure (A.4 - a), are not reflected in the edge attention weights, where node relationships are more variable. This suggests that the edge features provided the DyGAT model with a different perspective on node relationships. Nonetheless, the attention weights in time step 5 were very close to those in time step 6.

We can infer some general trends based on both types of attention weights. First, edge attention operates at a more granular level, focusing on specific connections, while node attention provides a broader overview of the entire graph, identifying important nodes. In addition, both edge and node attention exhibit dynamic patterns over time. However, the factors influencing this dynamic behavior differ. Edge attention is influenced by the changing wind speed and direction, while node attention is impacted by variations in meteorological and air quality features.

Overall, the attention weight heatmaps offer valuable insights into the spatial and temporal dependencies captured by DyGAT, providing a deeper understanding of the model's behavior and the underlying mechanisms affecting $PM_{2.5}$ levels. However, interpreting the exact meaning of attention weights can be challenging. Therefore, we combined attention analysis with evaluation metrics and visualizations that compare forecasted values to actual values to achieve a comprehensive analysis of the model's performance.

5.1.2 Spatiotemporal vs Temporal Forecasting

After training our DyGAT-Informer model and visualizing the results, we conducted comparisons between the DyGAT-Informer and other models. The first hypothesis of the study stated that integrating a model for capturing spatial dependencies with a time-series forecasting model in a hybrid architecture would improve $PM_{2.5}$ forecasts at each station rather than only using a time-series forecasting model. Therefore, the first comparison is between the performance of the DyGAT-Informer and the Informer alone to determine if DyGAT improved the forecasting results by accounting for the spatial dependencies between the stations. Table (6) presents the evaluation results for each individual station at 24-hour forecasting window. The evaluation results from Table (6), shows that incorporating the spatial dependencies between stations into the forecasting model

actually improved each individual station forecasts, which confirms the first study hypothesis.

Table 6

Evaluation Results of DyGAT-Informer vs. Informer for Beijing Stations (24-Hour Forecasting).

Station	DyGAT-Informer			Informer		
	MAE	RMSE	SMAPE	MAE	RMSE	SMAPE
Aotizhongxin	50.388	79.289	28.98%	55.242	83.320	32.04%
Changping	43.691	69.070	26.41%	47.426	72.017	28.55%
Dingling	43.264	66.642	26.97%	45.254	66.525	28.94%
Dongsi	56.685	89.270	31.01%	61.531	92.224	33.61%
Guanyuan	52.884	83.352	29.67%	56.959	85.639	32.28%
Gucheng	54.334	88.058	30.12%	58.039	91.960	32.08%
Huairou	42.410	65.566	25.46%	43.927	67.727	29.44%
Nongzhanguan	54.375	85.411	30.32%	59.196	89.083	33.15%
Shunyi	49.664	78.479	29.08%	53.987	83.154	32.3%
Tiantan	51.792	81.133	29.55%	55.769	83.295	31.74%
Wanliu	48.612	76.259	28.12%	52.590	79.158	31.71%
Wanshouxigong	57.125	91.082	30.75%	60.708	92.447	32.91%

5.1.3 Comparative Analysis with Baseline Models (Beijing Dataset)

The next step was to compare the performance of the DyGAT-Informer with the baseline models mentioned in chapter 4 “Experimental Design and Setup”. First, we trained the four baseline models we mentioned chapter 4 using Beijing dataset. Table (7) presents the evaluation results of all five models at three different forecasting windows, which are 24, 48, and 72 time-step and each time-step represents an hour according to the temporal resolution in the Beijing dataset.

After examining Table (7), the first observation is that our DyGAT-Informer outperformed all other models in all forecasting windows. Figure (9), shows the MAE, RMSE and SMAPE for all model over the forecasting windows. From the results, we found that the Informer model was the most effective at capturing the temporal dependencies in the Beijing dataset across both short-term and long-term forecasting

windows. This can be attributed to the Informer's ability to capture long-range temporal dependencies while simultaneously addressing localized patterns. The model's multi-scale representation capability, achieved through its Distilling Mechanism, enables it to hierarchically compress input sequences while preserving critical information. This allows the Informer to effectively capture short-term fluctuations in $PM_{2.5}$ levels, often influenced by meteorological conditions, while maintaining accurate long-term trends driven by seasonal cycles. Its ProbSparse self-attention further enhances this ability by focusing on the most informative dependencies across different temporal scales. As illustrated in Figure (7), this combination allows the Informer to balance short-term variability with long-term dynamics. In addition, the DyGAT's ability to capture the dynamic spatial dependencies between different locations, further enhanced the hybrid model's forecasts at each location.

Table 7

Evaluation Results for Baseline Models over Different Forecasting Windows (Beijing Dataset).

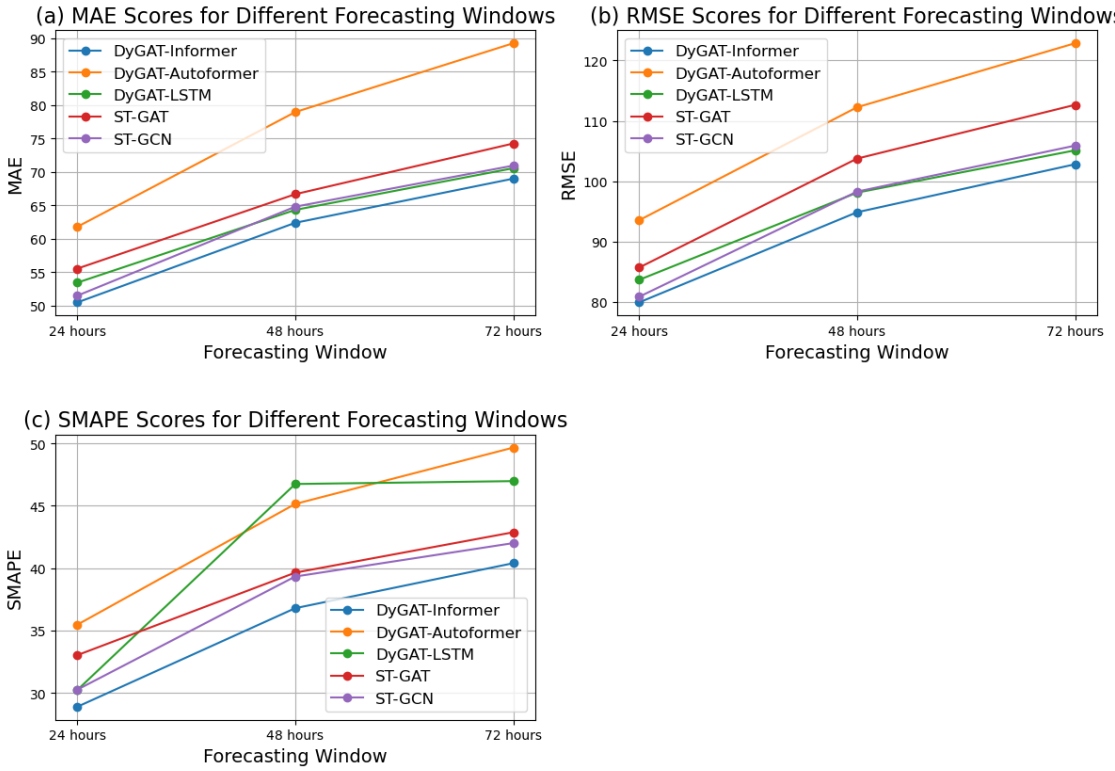
Model	24 Hours			48 Hours			72 Hours		
	MAE	RMSE	SMAPE	MAE	RMSE	SMAPE	MAE	RMSE	SMAPE
DyGAT-Informer	**50.43**	**79.9**	**28.88%**	**62.35**	**94.85**	**36.79%**	**68.96**	**102.8**	**40.4%**
DyGAT-Autoformer	61.74	93.51	35.46%	78.92	112.25	45.15%	89.22	122.86	49.67%
DyGAT-LSTM	53.39	83.64	30.22%	64.28	98.1	46.75%	70.52	105.11	46.98%
ST-GAT	55.49	85.67	33.02%	66.63	103.74	39.64%	74.22	112.65	42.88%
ST-GCN	51.44	80.83	30.25%	64.76	98.27	39.33%	70.91	105.89	42.01%

We also observed that the DyGAT-Autoformer lagged significantly behind all other models. The Autoformer was outperformed by the Informer in forecasting $PM_{2.5}$ levels due to fundamental differences in how these models handle the characteristics of volatile and non-stationary nature of the air quality data. $PM_{2.5}$ levels have high variability, often influenced by sudden environmental or human activities such as weather changes, traffic fluctuations, or industrial emissions. These abrupt changes make it difficult to identify stable seasonal or trend components, which are essential for models like Autoformer that utilizes decomposition techniques. The Autoformer's signal decomposition process is

designed to split time-series data into trend and seasonal components, a process that works well for datasets with clear, repeating patterns. However, in the case of $PM_{2.5}$ forecasting, the inherent instability and lack of predictable seasonality in the dataset diminish the effectiveness of this approach.

Figure 9

Evaluation Scores for the Models over Different Forecasting Windows for Beijing Dataset, (a) MAE, (b) RMSE, (c) SMAPE



Both ST-GAT and ST-GCN models use temporal convolution for temporal modeling but employ different GNN variants. ST-GCN demonstrates better performance with the Beijing dataset, even though standard GCNs are not dynamic GNNs like GATs. This result can be attributed to several factors. The ST-GCN and ST-GAT models have different architectural designs. ST-GCN utilizes spatiotemporal convolutions to capture both spatial and temporal dependencies simultaneously. On the other hand, ST-GAT focuses on attention-based spatial dependencies and then applies temporal processing. Depending on the dataset characteristics and model complexity, one architecture might be more effective at capturing the underlying patterns. Additionally, the hyperparameter

tuning process for ST-GAT may not have identified the optimal parameters, which could also impact performance.

When comparing DyGAT-Informer and ST-GAT, both models use graph attention networks to capture the spatial components in the data. However, the DyGAT-Informer exhibits a better performance than ST-GAT. A possible reason could be that air quality data have features like wind information that was part of the edge features we created which enabled our DyGAT to have a different perspective about the spatial dynamics, leading to better performance. Another difference between them is that our adjacency matrix was computed based on the nodes features using attention mechanism, while ST-GAT initialize the adjacency matrix as a learnable parameter. When we used learnable adjacency matrix in the DyGAT instead of attention based one at forecasting windows 24-hour, the MAE of the model have risen from 50.435 to 51.193, while the RMSE have risen from 79.907 to 80.757 and the SMAPE have risen from 28.88% to 29.55%.

Moreover, after examining the performance of DyGAT-LSTM model, it seems that the MAE and RMSE values are very close to ST-GCN at forecasting windows 48-hour and 72-hour, and at forecasting window 24-hour for the SMAPE. However, the SMAPE value for the DyGAT-LSTM was the highest among other models at 48-hour forecasting window and the second highest in the 72-Hour forecasting window. This could suggest that DyGAT-LSTM might have more difficulty in accurately predicting values that are small or close to zero, which is where SMAPE is more sensitive. This difficulty was more prominent at larger forecasting windows. LSTMs are powerful forecasting models and they are the most commonly used architectures in the literature related to time-series forecasting. However, simple LSTM-based architectures could struggle at long-range forecasting windows due to their vanishing gradient problem. Nevertheless, the seq2seq architecture that we have used with recursive training strategy can help LSTMs mitigate this issue.

To provide a visual perspective on the models' performances, a comparison of the true and predicted forecasts for each model is presented in Figure (A.5). For clarity, the plots for each model are shown separately, where one station (Aotizhongxin) was chosen for the plots as an example. After examining Figure (A.5), it seems that DyGAT-Informer, DyGAT-LSTM and ST-GCN closely follow the true values, with DyGAT-Informer

handling rapid variations particularly well. DyGAT-LSTM occasionally shows a slight lag during rapid changes, While ST-GCN seems to struggle with peak values.

DyGAT-Autoformer generally follows the trend but seems to be struggling with rapid changes compared to the other models, as it forecasts look smoother with less variations, this confirms that the Autoformer is less suitable for local and short-term trends. Finally, although ST-GAT is performing better than DyGAT-Autoformer as shown in the evaluation metrics, visually ST-GAT seems to perform less effectively in certain areas. It under estimate peaks values, and diverge more from the true values during periods of high fluctuations. In addition, it was the only model that have produced negative forecasts, which is not acceptable for $PM_{2.5}$ forecasting task, since $PM_{2.5}$ concentrations cannot be negative.

The discrepancies between the forecasts visualization and evaluation metrics results when it comes to comparing DyGAT-Autoformer and ST-GAT arise from how these evaluation tools capture different aspects of the model's performance. Metrics like MAE, RMSE, and SMAPE compute errors across all time steps. This means that localized issues such as under estimation of peak values or poor performance in fluctuations periods might be lost due to averaging the values over time. Therefore, even though ST-GAT tend to underestimate during high peaks and produce negative forecasts, its overall performance in capturing smaller values and responding to minor fluctuations was sufficient to reduce its average error. In contrast, the DyGAT-Autoformer forecasts were generally less responsive to the fluctuations in $PM_{2.5}$ levels.

5.2 Case Study - Nablus Dataset

The second part of our experiments were conducted using the Nablus dataset, to test the second hypothesis of the study. This dataset is very different from the Beijing dataset. The first difference is in dataset size, where each station in the Beijing dataset have 35064 time-step, while in the Nablus dataset each station has only 2692 time-step. Another difference is that the meteorological data for Nablus dataset is the same for all 8 stations, the reasons is that all stations are located within a small area in the same city, therefore they are very close to each other. The small distances between stations in Nablus could mean that the spatial interactions between different pairs of nodes may not be as diverse as in the Beijing dataset. However, other factors like the topography of the area could

have an influence on the spatial relations between the nodes even in a relatively small area. The last difference is the temporal resolution of the data, unlike the Beijing dataset that have a temporal resolution of 1 hour, the Nablus dataset's temporal resolution is 30 minutes.

Initially, transfer learning was considered, where the pre-trained model from the Beijing dataset would be fine-tuned with the Nablus dataset. However, we encountered significant challenges. The pre-trained model was specifically designed to handle a diverse set of pollutants, meteorological features, and edge features unique to the Beijing dataset, and it could not be seamlessly transferred to the Nablus dataset. We attempted to adapt the model by aligning its architecture and parameters with the new dataset's features but faced issues such as mismatches in feature characteristics. These included uniform weather conditions across stations, different set of meteorological data, and the absence of certain pollutants present in the Beijing dataset.

Additionally, edge features in the Beijing dataset contains both static and temporal information, while the Nablus dataset only includes one static feature, which is the distance between stations. Another challenge involved incorporating additional GIS features to test the second study hypothesis. To effectively address the second hypothesis of our study, which states that integrating additional information about nearby pollution sources will enhance the forecasting accuracy, it is essential to train a version of the model specifically tailored to the second dataset.

First, we performed hyper-parameter tuning to the models using the Nablus dataset. In the beginning, we used the same forecasting windows used with the Beijing dataset. However, the small size of the dataset drastically affected the models' performance. Therefore, we chose smaller forecasting windows to train and test the model, and they are 6 time-steps (3 Hours), 12 time-steps (6 hours) and 24 time-steps (12 hours).

5.2.1 Impact of Contextual Features on Forecasting Accuracy

During the hyper-parameter tuning phase, we found that the DyGAT-LSTM model performed better than DyGAT-Informer, which was expected due to the small dataset size. The Informer model struggled to effectively learn from the limited data. Therefore, we chose the DyGAT-LSTM hybrid model to test the second hypothesis of the study. We

hypothesized that adding additional context about nearby pollution sources like distance and direction from the measuring station to the source could help the model to produce better forecasts. These additional features are obtained using GIS. We trained the DyGAT-LSTM with the additional GIS data from Saleh et.al. [55] study, and then we trained the model without the additional features.

Additionally, we trained the LSTM models without our DyGAT model, one with GIS data and one without them. This approach was to test the first study hypothesis, and to examine whether the DyGAT model could capture spatial dependencies in a small study area where the nodes are very close to each other, and the weather features are the same across all nodes. The results are presented in Table (8), where the superscript letter 'G' indicates that the model was trained with the additional GIS, and the superscript letters 'NG' indicates that the model was trained without them.

Table 8

Evaluation Results of DyGAT-LSTM and LSTM Models with and without GIS Data (Nablus Dataset)

Model / Data Features	MAE	RMSE	SMAPE %
$LSTM^{G}$	6.171	10.646	27%
$LSTM^{NG}$	6.262	10.685	27.24%
DyGAT-$LSTM^{G}$	**5.978**	**10.488**	**26.01%**
DyGAT-$LSTM^{NG}$	6.105	10.612	26.53%

Form the results presented in Table (8), the DyGAT-$LSTM^{G}$ model showed better performance than the other models. The results were close to each other, so to determine whether the performance enhancement from incorporating DyGAT and the additional GIS features is significant, we conducted paired t-tests [82] on the MAE, RMSE, and SMAPE values for the four models.

The paired t-test evaluates whether the mean differences in evaluation metrics between models are statistically significant. By comparing the observed differences in performance across pairs of models, the test assesses whether improvements in MAE, RMSE, and SMAPE are due to actual changes or if they occurred by chance. The paired t-test relies on two key statistics: the t-statistic and the p-value.

The t-statistic measures the difference between sample means relative to the variability within the sample, indicating the extent to which the observed difference is significant. The p-value, on the other hand, measures the likelihood of observing a difference at least as large as the one found if there were actually no real difference (the null hypothesis). In other words, it assesses whether the observed result is unexpected under the assumption of no real effect. A low p-value suggests that the result is unusual under the null hypothesis, indicating that the observed difference is likely significant. The threshold for significance is commonly set at 0.05. This means that if the p-value is less than 0.05, the result is considered statistically significant, suggesting that the observed difference is unlikely to have occurred by chance. The results of the paired t-tests are presented in Table (9).

Table 9

Statistical Significance of Performance Differences between Models (Nablus Dataset)

Comparison	MAE			RMSE			SMAPE		
	t-statistic	p-value	Significant?	t-statistic	p-value	Significant?	t-statistic	p-value	Significant?
$LSTM^{G}$ vs DyGAT-$LSTM^{G}$	10.957	0.008	Yes	16.403	0.003	Yes	6.739	0.021	Yes
$LSTM^{G}$ vs DyGAT-$LSTM^{NG}$	5.508	0.031	Yes	10.022	0.009	Yes	2.181	0.160	No
$LSTM^{G}$ vs $LSTM^{NG}$	-19.065	0.002	Yes	-2.725	0.112	No	-1.363	0.305	No
DyGAT-$LSTM^{G}$ vs DyGAT-$LSTM^{NG}$	-7.199	0.018	Yes	-9.760	0.01	Yes	-2.809	0.106	No
DyGAT-$LSTM^{G}$ vs $LSTM^{NG}$	-17.790	0.003	Yes	-19.782	0.002	Yes	-27.919	0.001	Yes
DyGAT-$LSTM^{NG}$ vs $LSTM^{NG}$	-22.289	0.002	Yes	-4.719	0.042	Yes	-4.401	0.047	Yes

The first comparison between $LSTM^{G}$ and DyGAT-$LSTM^{G}$ shows significant differences across all three metrics. This suggests that DyGAT-$LSTM^{G}$ performs better than the $LSTM^{G}$ in forecasting $PM_{2.5}$ levels. The high t-statistics for these comparisons (10.957,

16.403, and 6.739 respectively) further confirm that the improvements are substantial and not due to chance. In the second comparison between $LSTM^{G}$ and DyGAT-$LSTM^{NG}$, the results show a mix of significant and non-significant differences. While both MAE and RMSE show statistically significant differences ($p = 0.031$ and $p = 0.009$), indicating that the DyGAT-$LSTM^{NG}$ outperforms the $LSTM^{G}$ in these metrics, the SMAPE comparison ($p = 0.160$) is not significant. This suggests that the two models may perform similarly in terms of percentage error, and the absence of GIS data in the DyGAT-LSTM may not have a major impact on this metric.

When comparing $LSTM^{G}$ to $LSTM^{NG}$, the tests show there are no significant differences for RMSE and SMAPE. However, the MAE comparison is significant ($p = 0.002$). This suggests that the $LSTM^{G}$ model is significantly better in terms of absolute error reduction, but its performance on RMSE and SMAPE may not differ substantially from the version without GIS data. The fourth comparison, between DyGAT-$LSTM^{G}$ and DyGAT-$LSTM^{NG}$, highlights a significant improvement in the version with GIS data for MAE and RMSE. The absence of a significant difference in SMAPE is similar to the results in the second comparison, suggesting that GIS data has a more effect on absolute error and squared error, rather than percentage error.

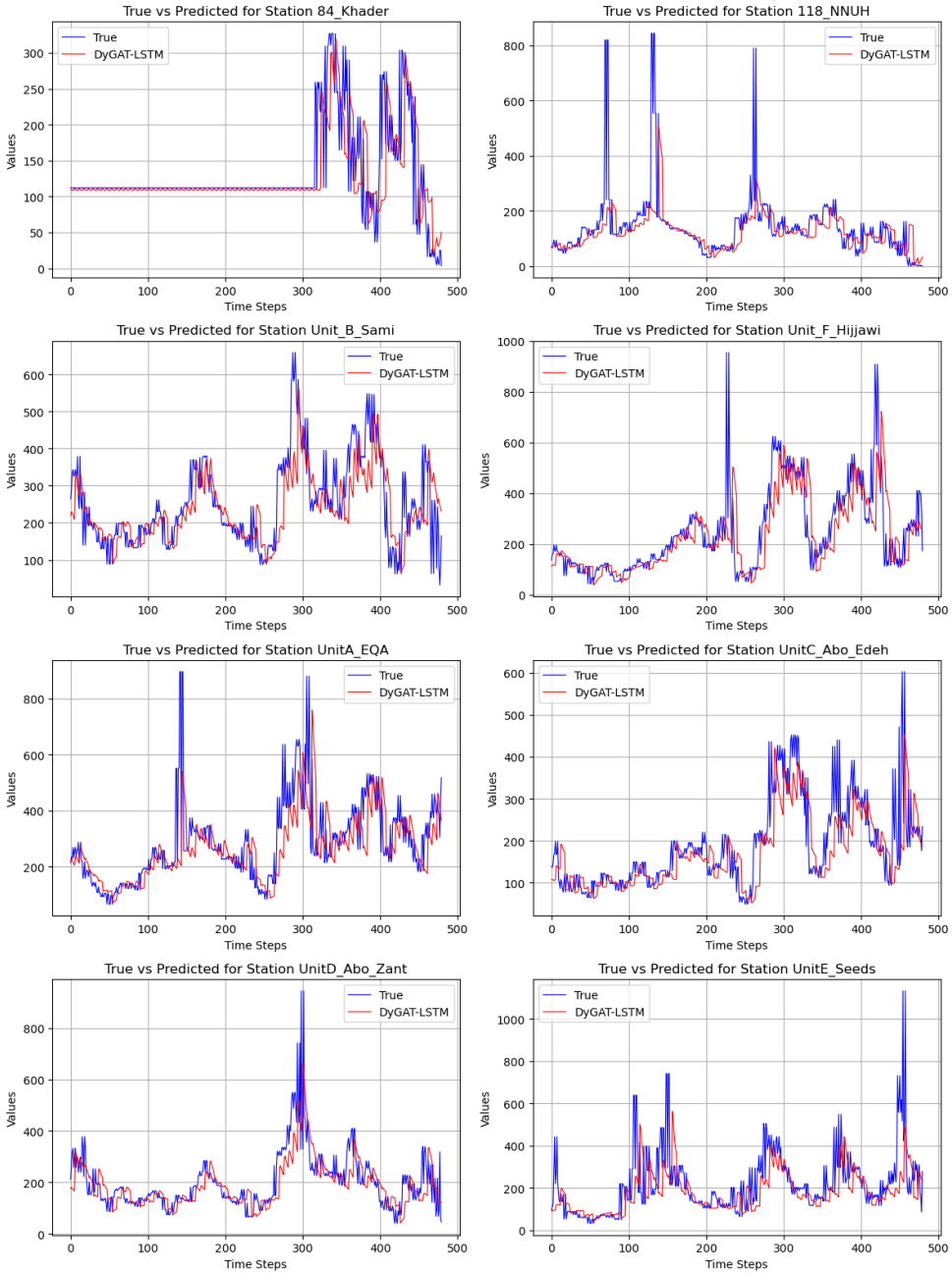
In the comparison between DyGAT-$LSTM^{G}$ and $LSTM^{NG}$, the p-values for all three metrics are significant, indicating that the DyGAT-$LSTM^{G}$ significantly outperforms $LSTM^{NG}$ in all metrics. The extremely high t-statistics, particularly in the case of SMAPE (-27.919), reflect the large performance gap between these two models, emphasizing the positive effect of both using the DyGAT model and the inclusion of GIS data in improving forecasting accuracy. Finally, the comparison between DyGAT-$LSTM^{NG}$ and $LSTM^{NG}$ also shows significant differences across all three metrics, with p-values below 0.05 in each case. This suggests that even without GIS data, the DyGAT-LSTM model is consistently superior to the LSTM model, reinforcing the effectiveness of the DyGAT model for spatiotemporal forecasting tasks.

Despite challenges such as a small dataset, the close proximity of the stations, and identical meteorological data across stations, the DyGAT-$LSTM^{G}$ was able to enhance the performance according to results in Table 9. The reason for this is that the $PM_{2.5}$ readings at each station is not only influenced by the weather, but also by other factors

like the topography of the place. Geographic features like mountains and valleys can influence the dispersion and concentration of $PM_{2.5}$. Therefore, the DyGAT was able to detect these factors from $PM_{2.5}$ readings even though they are not explicitly present in the nodes features. The second observation from the results is that incorporating GIS data on nearby pollution sources improved the model performance. This result confirms the second hypothesis of the study. Figure (10) presents a comparison between the predicted values from DyGAT-LSTM^{G} and the actual values at each station in Nablus city.

Figure 10

Actual vs Predicted Values by DyGAT-LSTM(GIS) Model (Nablus Dataset)



For most stations, the predicted lines in figure (10) closely follow the true lines except at certain peak values that occurred suddenly. However, at stations ‘Unit_B_Sami’ and '84_Khader', the model performed better at high $PM_{2.5}$ levels. Moreover, the model was able forecast the minor fluctuations in the $PM_{2.5}$ levels.

5.2.2 Comparative Analysis with Baseline Models (Nablus Dataset)

The next step was training the baseline models using the Nablus dataset where the GIS features were included. Then, the models performances were evaluated with forecasting windows of 6, 12, and 24 time-steps, which correspond to 3-hours, 6-hours and 12-hours forecasting windows respectively. The results for all models are presented in Table (10), and the evaluation metrics visualizations are shown in Figure (11).

Table 10

Evaluation results of Baseline Models (Nablus dataset).

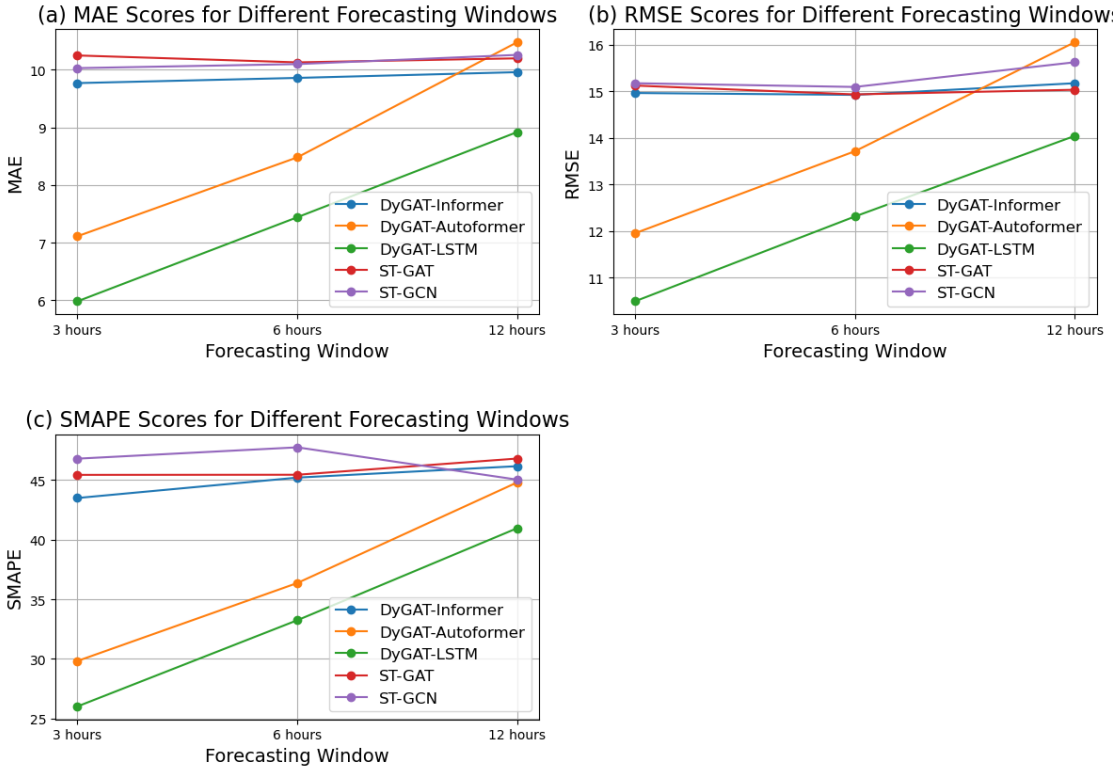
Model	3 Hours			6 Hours			12 Hours		
	MAE	RMSE	SMAPE	MAE	RMSE	SMAPE	MAE	RMSE	SMAPE
DyGAT-Informer	9.77	14.96	43.49%	9.86	14.92	45.20%	9.96	15.17	46.16%
DyGAT-Autoformer	7.11	11.95	29.82%	8.48	13.71	36.36%	10.48	16.05	44.80%
DyGAT-LSTM	5.98	10.49	26.01%	7.44	12.31	33.24%	8.92	14.04	40.96%
ST-GAT	10.25	15.12	45.43%	10.13	14.93	45.44%	10.20	15.03	46.80%
ST-GCN	10.03	15.17	46.79%	10.10	15.09	47.74%	10.26	15.62	45.03%

After examining Table (10) and Figure (11), the first observation is that the DyGAT-LSTM model outperformed all other models. It was expected that using LSTM instead of Informer would result in better forecasts since the data size is small. Another reason is that the forecasting windows are smaller than the ones used with Beijing dataset, which alleviated the LSTM struggles with longer forecasting windows.

The second observation is that the DyGAT-Autoformer is the second-best performing model for forecasting windows of 3-hours and 6-hours across all metrics, it also ranks second for the 12-hour forecasting window when evaluated using the SMAPE metric. Even though DyGAT-Autoformer was the worst-performing model when used with the Beijing dataset, it demonstrated better performance than other models with the smaller Nablus dataset, despite its sophisticated architecture, which typically requires large amounts of data. A possible explanation is that Autoformer uses autocorrelation attention rather than traditional self-attention. Autocorrelation attention focuses on capturing dependencies based on the periodicity of the time series, allowing it to aggregate information at a series level. This approach can be more efficient compared to the point-wise aggregation used in self-attention, especially when dealing with smaller datasets. As a result, Autoformer was able to outperform the Informer model with the smaller Nablus dataset.

Figure 11

Evaluation Scores for the Models over Different Forecasting Windows for Nablus Dataset, (a) MAE, (b) RMSE, (c) SMAPE



Finally, the DyGAT-Informer, ST-GAT and ST-GCN seem to have similar performances that do not change much over different forecasting windows, which suggested that the models were not learning from data. After examining Figure (A.6) in Appendix A, which visualize the actual values and predicted values, the predictions from all three models appear to be almost flat with some periodic oscillations. This indicates that the models might be under-fitting the data, failing to capture the actual trends and fluctuations in $PM_{2.5}$ levels.

The periodic oscillations could stem from factors like model complexity. If the model is too complex, it could be overfitting the training data, capturing noise or irrelevant patterns, which is presented as oscillations in the forecasts. Through the hyper-parameter tuning phase, we tried different parameters for number of layers, and hidden units sizes, but the performance of these three models remained the same.

When it comes to the DyGAT-Informer, another possible cause for oscillations could be the use of ‘Fixed’ temporal embeddings. Fixed embedding method was used to create temporal embeddings for the transformer based models and LSTM model. This method is based on sine and cosine functions, and designed to capture the periodicity in the data, which is useful in many temporal tasks. However, this type of embedding could have contributed to the oscillatory behavior seen in the DyGAT-Informer, especially if the model relied on these periodic signals and failed to capture more complex, non-periodic trends in the data.

To investigate whether the fixed temporal embeddings were responsible for the observed oscillation patterns in the three models, we retrained the DyGAT-Informer using the 'Learned' embedding method (discussed in Chapter 3). The results are presented in Figure (A.7). After analyzing the figure, it became clear that the oscillations persisted but with altered frequencies and shapes. This indicates that the ‘Fixed’ temporal embeddings are not the cause of the oscillatory behavior. In conclusion, it seems that the DyGAT-Informer, ST-GAT and ST-GCN are not suitable for the Nablus dataset.

Finally, both DyGAT-LSTM and DyGAT-Autoformer seem to learn the trends in the data, with DyGAT-LSTM being the best performing model overall. To confirm that DyGAT-Autoformer is also learning the patterns and trends, we visualized its forecasts alongside DyGAT-LSTM’s predictions for a quick visual comparison. The visualization

is presented in Figure (A.8), where both models generally demonstrate good accuracy in predicting the true values across most stations. However, it seems that DyGAT-Autoformer struggles slightly more in capturing the underlying trends in the data.

It is important to note that while DyGAT-LSTM outperformed all other models on the Nablus dataset, this result might not necessarily be generalized if the dataset spanned a longer time period or contained more varied meteorological data across stations. The Nablus dataset consists of only 2,692 records per station (21536 record in total) over a span of two months, compared to the Beijing dataset, which includes 35,064 records per station (420768 record in total) over four years. The shorter time span and identical meteorological conditions across stations in the Nablus dataset may have favored simpler architectures like LSTM that excel with smaller and less complex datasets. In contrast, a model like DyGAT-Informer, which was designed to leverage larger datasets and capture long-term patterns, might perform better if more records were available. This observation highlights the importance of dataset characteristics in determining model performance and underscores the need for further experimentation with expanded datasets to validate these conclusions.

Chapter Six
Conclusions and Future Work

In this chapter, we present conclusions drawn from the experiment results and the analysis we provided in the previous chapter. We will also outline potential directions for future work, suggesting how the study can be extended and improved.

6.1 Conclusions

In this study, we created a dynamic Graph Attention Network DyGAT to capture the dynamic spatial relations between different nodes in a graph. Then, we combined DyGAT with the Informer a time-series Transformer in a hybrid framework to use it for $PM_{2.5}$ forecasting. We trained DyGAT-Informer using the Beijing Multi-site Air Quality Dataset. We then compared its performance with a version of the model that only uses the Informer to assess the impact of modeling spatial dependencies between stations on forecasting accuracy. The DyGAT model demonstrated its ability to capture dynamic spatial relations between nodes by leveraging engineered features, such as wind-related information, resulting in improved forecasts compared to the Informer alone. This was evidenced by better MAE, RMSE, and SMAPE results across all stations.

Moreover, we presented a case study using an air quality dataset from Nablus city in Palestine, where we included contextual information about nearby sources of pollution. The results of this case study confirms that adding other contextual features in addition to historical $PM_{2.5}$ and meteorological data can indeed improve the forecasts. In this case study, we also confirmed the ability of our DyGAT model to capture spatial dependencies even in small study area, and homogenous weather data.

6.2 Future Work

While this study demonstrated the efficacy of the DyGAT-Informer model on the Beijing Dataset, further research could involve adapting the model to a larger and more diverse datasets. This could include regions with different climate conditions or varying levels of pollution to assess the model's generalizability and robustness. In addition, we can explore extending our hybrid model to incorporate other data sources that could enhance the air quality forecasting. Potential sources include satellite images, traffic-related data, and information about nearby pollution sources as demonstrated in the case study.

Another possible direction for future work is modifying the model to produce real-time forecasting capabilities by providing the model with real-time data streams and creating forecasts that provide insights for policymakers and public health officials. We can also test the DyGAT-Informer is other spatiotemporal forecasting tasks such as traffic forecasting, or wind speed forecasting.

List of Abbreviations

Abbreviation	Meaning
ANN	Artificial Neural Network
AOD	Aerosol Optical Depth
CNN	Convolutional Neural Networks
CTMs	Chemical Transport Models
CV	Coefficient of Variation
DL	Deep Learning
DBN	Deep Belief Network
EMD	Empirical Mode Decomposition
FCN	Fully Connected Network
FEA	Frequency Enhanced Attention
GAT	Graph Attention Network
GCN	Graph Convolutional Network
GIS	Geographic Information Systems
GNN	Graph Neural Networks
GRU	Gated Recurrent Network
GTCN	Gated Temporal Convolutional Layer
IQR	Interquartile Range
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
ML	Machine Learning
MLP	Multi-Layer Perceptron
NLP	Natural Language Processing
PM	Particulate Matter
RF	Random Forests
RMSE	Root Mean Square Error
RNN	Recurrent Neural Networks
Seq2Seq	Sequence-To-Sequence
SMAPE	Symmetric Mean Absolute Percentage Error
SVM	Support Vector Machine
VMD	Variational Mode Decomposition
WHO	World Health Organization

References

[1] World Health Organization, “Ambient (outdoor) air pollution.” Accessed: Aug. 05, 2023. [Online]. Available: https://tinyurl.com/ambient-air-quality-health

[2] G. D. Thurston, “Outdoor Air Pollution: Sources, Atmospheric Transport, and Human Health Effects,” in *International Encyclopedia of Public Health*, Elsevier, 2017, pp. 367–377. doi: 10.1016/B978-0-12-803678-5.00320-9.

[3] M. C. Turner *et al.*, “Outdoor air pollution and cancer: An overview of the current evidence and public health recommendations,” *CA Cancer J Clin*, vol. 70, no. 6, pp. 460–479, Nov. 2020, doi: 10.3322/caac.21632.

[4] R. El Morabet, “Effects of Outdoor Air Pollution on Human Health,” in *Reference Module in Earth Systems and Environmental Sciences*, Elsevier, 2018. doi: 10.1016/B978-0-12-409548-9.11012-7.

[5] IQAir, “IQAir 2023 World Air Quality Report,” 2023. [Online]. Available: https://www.iqair.com/dl/2023_World_Air_Quality_Report.pdf?srsltid=AfmBOooBKsXzzfh3a1kL8Sng0fAur2ZFUYKBqgzq7EH842sBwePNQ-ie

[6] T.-M. Chen, W. G. Kuschner, S. Shofer, and J. Gokhale, “Outdoor Air Pollution: Overview and Historical Perspective,” *Am J Med Sci*, vol. 333, no. 4, pp. 230–234, Apr. 2007, doi: 10.1097/MAJ.0b013e31803b8c91.

[7] M. Méndez, M. G. Merayo, and M. Núñez, “Machine learning algorithms to forecast air quality: a survey,” *Artif Intell Rev*, Sep. 2023, doi: 10.1007/s10462-023-10424-4.

[8] H. Hajmohammadi and B. Heydecker, “Multivariate time series modelling for urban air quality,” *Urban Clim*, vol. 37, May 2021, doi: 10.1016/j.uclim.2021.100834.

[9] Q. Liao, M. Zhu, L. Wu, X. Pan, X. Tang, and Z. Wang, “Deep Learning for Air Quality Forecasts: a Review,” *Curr Pollut Rep*, vol. 6, no. 4, pp. 399–409, Dec. 2020, doi: 10.1007/s40726-020-00159-z.

[10] S. Abdullah, M. Ismail, A. N. Ahmed, and W. N. W. Mansor, “Big data analytics and artificial intelligence in air pollution studies for the prediction of particulate matter concentration,” in *Proceedings of the 3rd International Conference on Telecommunications and Communication Engineering*, New York, NY, USA: ACM, Nov. 2019, pp. 90–94. doi: 10.1145/3369555.3369557.

[11] A. Sotomayor-Olmedo *et al.*, “Evaluating trends of airborne contaminants by using support vector regression techniques,” *CONIELECOMP 2011, 21st International Conference on Electrical Communications and Computers*, pp. 137–141, 2011.

[12] B. Y. Kim, Y. K. Lim, and J. W. Cha, “Short-term prediction of particulate matter (PM10 and PM2.5) in Seoul, South Korea using tree-based machine learning

algorithms," *Atmos Pollut Res*, vol. 13, no. 10, Oct. 2022, doi: 10.1016/j.apr.2022.101547.

[13] L. Yao, N. Lu, and S. Jiang, "Artificial neural network (ANN) for multi-source PM2.5 estimation using surface, MODIS, and meteorological data," in *Proceedings - 2012 International Conference on Biomedical Engineering and Biotechnology, iCBEB 2012*, 2012, pp. 1228–1231. doi: 10.1109/iCBEB.2012.81.

[14] S. Agarwal *et al.*, "Air quality forecasting using artificial neural networks with real time dynamic error correction in highly polluted regions," *Science of the Total Environment*, vol. 735, Sep. 2020, doi: 10.1016/j.scitotenv.2020.139454.

[15] K. I. Hoi, K. V. Yuen, and K. M. Mok, "Improvement of the multilayer perceptron for air quality modelling through an adaptive learning scheme," *Comput Geosci*, vol. 59, pp. 148–155, 2013, doi: 10.1016/j.cageo.2013.06.002.

[16] M. Fu, W. Wang, Z. Le, and M. S. Khorram, "Prediction of particular matter concentrations by developed feed-forward neural network with rolling mechanism and gray model," *Neural Comput Appl*, vol. 26, no. 8, pp. 1789–1797, Nov. 2015, doi: 10.1007/s00521-015-1853-8.

[17] T. Thanavanich, M. Yaibuates, and P. Suchaya, "Improving the Accuracy of Forecasting PM2.5 Concentrations with Hybrid Neural Network Model," in *2021 Joint 6th International Conference on Digital Arts, Media and Technology with 4th ECTI Northern Section Conference on Electrical, Electronics, Computer and Telecommunication Engineering, ECTI DAMT and NCON 2021*, Institute of Electrical and Electronics Engineers Inc., Mar. 2021, pp. 18–22. doi: 10.1109/ECTIDAMTNCON51128.2021.9425724.

[18] Y. Zhou, F. J. Chang, L. C. Chang, I. F. Kao, and Y. S. Wang, "Explore a deep learning multi-output neural network for regional multi-step-ahead air quality forecasts," *J Clean Prod*, vol. 209, pp. 134–145, Feb. 2019, doi: 10.1016/j.jclepro.2018.10.243.

[19] X. Song, J. Huang, and D. Song, "Air Quality Prediction based on LSTM-Kalman Model," in *2019 IEEE 8th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, 2019, pp. 695–699.

[20] D. Seng, Q. Zhang, X. Zhang, G. Chen, and X. Chen, "Spatiotemporal prediction of air quality based on LSTM neural network," *Alexandria Engineering Journal*, vol. 60, no. 2, pp. 2021–2032, Apr. 2021, doi: 10.1016/j.aej.2020.12.009.

[21] D. Madaan, R. Dua, P. Mukherjee, and B. Lall, "VayuAnukulani: Adaptive Memory Networks for Air Pollution Forecasting," in *2019 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, 2019, pp. 1–5.

[22] J. Ma, J. C. P. Cheng, C. Lin, Y. Tan, and J. Zhang, "Improving air quality prediction accuracy at larger temporal resolutions using deep learning and transfer learning techniques," *Atmos Environ*, vol. 214, Oct. 2019, doi: 10.1016/j.atmosenv.2019.116885.

[23] J. Ma, Z. Li, J. C. P. Cheng, Y. Ding, C. Lin, and Z. Xu, "Air quality prediction at new stations using spatially transferred bi-directional long short-term memory network," *Science of the Total Environment*, vol. 705, Feb. 2020, doi: 10.1016/j.scitotenv.2019.135771.

[24] J. Ma, Y. Ding, V. J. L. Gan, C. Lin, and Z. Wan, "Spatiotemporal Prediction of PM2.5 Concentrations at Different Time Granularities Using IDW-BLSTM," *IEEE Access*, vol. 7, pp. 107897–107907, 2019, doi: 10.1109/ACCESS.2019.2932445.

[25] L. Zhang, P. Liu, L. Zhao, G. Wang, W. Zhang, and J. Liu, "Air quality predictions with a semi-supervised bidirectional LSTM neural network," *Atmos Pollut Res*, vol. 12, no. 1, pp. 328–339, Jan. 2021, doi: 10.1016/j.apr.2020.09.003.

[26] T.-C. Bui, V.-D. Le, and S. K. Cha, "A Deep Learning Approach for Forecasting Air Pollution in South Korea Using LSTM," *ArXiv*, 2018.

[27] M. H. Nguyen, P. Le Nguyen, K. Nguyen, V. A. Le, T. H. Nguyen, and Y. Ji, "PM2.5 Prediction Using Genetic Algorithm-Based Feature Selection and Encoder-Decoder Model," *IEEE Access*, vol. 9, pp. 57338–57350, 2021, doi: 10.1109/ACCESS.2021.3072280.

[28] T. Li, M. Hua, and X. Wu, "A Hybrid CNN-LSTM Model for Forecasting Particulate Matter (PM2.5)," *IEEE Access*, vol. 8, pp. 26933–26940, 2020, doi: 10.1109/ACCESS.2020.2971348.

[29] D. Qin, J. Yu, G. Zou, R. Yong, Q. Zhao, and B. Zhang, "A Novel Combined Prediction Scheme Based on CNN and LSTM for Urban PM2.5 Concentration," *IEEE Access*, vol. 7, pp. 20050–20059, 2019, doi: 10.1109/ACCESS.2019.2897028.

[30] V. D. Le, T. C. Bui, and S. K. Cha, "Spatiotemporal deep learning model for citywide air pollution interpolation and prediction," in *Proceedings - 2020 IEEE International Conference on Big Data and Smart Computing, BigComp 2020*, Institute of Electrical and Electronics Engineers Inc., Feb. 2020, pp. 55–62. doi: 10.1109/BigComp48618.2020.00-99.

[31] Z. Zhang, Y. Zeng, and K. Yan, "A hybrid deep learning technology for PM2.5 air quality forecasting," *Environmental Science and Pollution Research*, vol. 28, no. 29, pp. 39409–39422, Aug. 2021, doi: 10.1007/s11356-021-12657-8.

[32] Y. Qi, Q. Li, H. Karimian, and D. Liu, "A hybrid model for spatiotemporal forecasting of PM 2.5 based on graph convolutional neural network and long short-term memory," *Science of the Total Environment*, vol. 664, pp. 1–10, May 2019, doi: 10.1016/j.scitotenv.2019.01.333.

[33] M. Teng *et al.*, "72-hour real-time forecasting of ambient PM2.5 by hybrid graph deep neural network with aggregated neighborhood spatiotemporal information," *Environ Int*, vol. 176, p. 107971, Jun. 2023, doi: 10.1016/j.envint.2023.107971.

[34] W. Mao, L. Jiao, W. Wang, J. Wang, X. Tong, and S. Zhao, “A hybrid integrated deep learning model for predicting various air pollutants,” *GIsci Remote Sens*, vol. 58, no. 8, pp. 1395–1412, Nov. 2021, doi: 10.1080/15481603.2021.1988429.

[35] L. Gao, M. Liao, and D. Zhang, “Multi-site air quality prediction based on graph convolutional neural network-bi-directional LSTM model,” in *Fifth International Conference on Computer Information Science and Artificial Intelligence (CISAI 2022)*, Y. Zhong, Ed., SPIE, Mar. 2023, p. 101. doi: 10.1117/12.2667705.

[36] S. Wang, Y. Li, J. Zhang, Q. Meng, L. Meng, and F. Gao, “PM2.5-GNN: A Domain Knowledge Enhanced Graph Neural Network for PM2.5 Forecasting,” in *GIS: Proceedings of the ACM International Symposium on Advances in Geographic Information Systems*, Association for Computing Machinery, Nov. 2020, pp. 163–166. doi: 10.1145/3397536.3422208.

[37] K. Zhang, X. Zhang, H. Song, H. Pan, and B. Wang, “Air Quality Prediction Model Based on Spatiotemporal Data Analysis and Metalearning,” *Wirel Commun Mob Comput*, vol. 2021, 2021, doi: 10.1155/2021/9627776.

[38] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, “Graph Attention Networks,” in *International Conference on Learning Representations*, Oct. 2018. [Online]. Available: http://arxiv.org/abs/1710.10903

[39] S. Wang, L. Qiao, W. Fang, G. Jing, V. S. Sheng, and Y. Zhang, “Air Pollution Prediction Via Graph Attention Network and Gated Recurrent Unit,” *Computers, Materials and Continua*, vol. 73, no. 1, pp. 673–687, 2022, doi: 10.32604/cmc.2022.028411.

[40] H. Zhou, F. Zhang, Z. Du, and R. Liu, “Forecasting PM2.5 using hybrid graph convolution-based model considering dynamic wind-field to offer the benefit of spatial interpretability,” *Environmental Pollution*, vol. 273, Mar. 2021, doi: 10.1016/j.envpol.2021.116473.

[41] J. Li, X. Shao, and R. Sun, “A DBN-based deep neural network model with multitask learning for online air quality prediction,” *Journal of Control Science and Engineering*, vol. 2019, 2019, doi: 10.1155/2019/5304535.

[42] Y. Li, X. Shen, D. Han, J. Sun, and Y. Shen, “Spatio-temporal-Aware Sparse Denoising Autoencoder Neural Network for Air Quality Prediction,” in *2018 5th IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS)*, IEEE, 2018, pp. 96–100.

[43] A. Vaswani *et al.*, “Attention is all you need,” in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, in NIPS’17. Red Hook, NY, USA: Curran Associates Inc., 2017, pp. 6000–6010.

[44] Y. Liang *et al.*, “AirFormer: Predicting Nationwide Air Quality in China with Transformers,” *ArXiv*, no. 2211.15979, 2023, [Online]. Available: www.aaai.org

[45] M. A. A. Al-qaness *et al.*, “ResInformer: Residual Transformer-Based Artificial Time-Series Forecasting Model for PM2.5 Concentration in Three Major Chinese Cities,” *Mathematics*, vol. 11, no. 2, Jan. 2023, doi: 10.3390/math11020476.

[46] H. Zhou *et al.*, “Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 12, pp. 11106–11115, May 2021, doi: 10.1609/aaai.v35i12.17325.

[47] Z. Ma *et al.*, “Spatial and temporal characteristics analysis and prediction model of PM2.5 concentration based on SpatioTemporal-Informer model,” *PLoS One*, vol. 18, no. 6, p. e0287423, Jun. 2023, doi: 10.1371/journal.pone.0287423.

[48] H. Wang, L. Zhang, and R. Wu, “MSAFormer: A Transformer-Based Model for PM2.5 Prediction Leveraging Sparse Autoencoding of Multi-Site Meteorological Features in Urban Areas,” *Atmosphere (Basel)*, vol. 14, no. 8, Aug. 2023, doi: 10.3390/atmos14081294.

[49] Z. Zhang and S. Zhang, “Modeling air quality PM2.5 forecasting using deep sparse attention-based transformer networks,” *International Journal of Environmental Science and Technology*, 2023, doi: 10.1007/s13762-023-04900-1.

[50] P. Li, T. Zhang, and Y. Jin, “A Spatio-Temporal Graph Convolutional Network for Air Quality Prediction,” *Sustainability (Switzerland)*, vol. 15, no. 9, May 2023, doi: 10.3390/su15097624.

[51] S. Yun, M. Jeong, R. Kim, J. Kang, and H. J. Kim, “Graph transformer networks,” *Adv Neural Inf Process Syst*, vol. 32, 2019.

[52] Y. Li and J. M. F. Moura, “Forecaster: A Graph Transformer for Forecasting Spatial and Time-Dependent Data,” *arXiv preprint arXiv:1909.04019*, Sep. 2019, doi: 10.3233/FAIA200231.

[53] Z. Zhang, S. Zhang, X. Zhao, L. Chen, and J. Yao, “Temporal Difference-based Graph Transformer Networks for Air Quality PM2.5 Prediction: A Case Study in China,” *Front Environ Sci*, vol. 10, no. 2296–665X, 2022, doi: 10.21203/rs.3.rs-1168251/v1.

[54] Song Chen, “Beijing Multi-Site Air Quality,” UCI Machine Learning Repository.

[55] T. Saleh and A. Khader, “Urban Particulate Matter Hazard Mapping and Monitoring Site Selection in Nablus, Palestine,” *Atmosphere (Basel)*, vol. 13, no. 7, p. 1134, Jul. 2022, doi: 10.3390/atmos13071134.

[56] Z. Niu, G. Zhong, and H. Yu, “A review on the attention mechanism of deep learning,” *Neurocomputing*, vol. 452, pp. 48–62, Sep. 2021, doi: 10.1016/j.neucom.2021.03.091.

[57] G. Brauwers and F. Frasincar, “A General Survey on Attention Mechanisms in Deep Learning,” *IEEE Trans Knowl Data Eng*, vol. 35, no. 4, pp. 3279–3298, Apr. 2023, doi: 10.1109/TKDE.2021.3126456.

[58] D. Soydaner, “Attention mechanism in neural networks: where it comes and where it goes,” *Neural Comput Appl*, vol. 34, no. 16, pp. 13371--13385, 2022.

[59] C. Sun *et al.*, “Attention-based graph neural networks: a survey,” *Artif Intell Rev*, vol. 56, pp. 2263–2310, Nov. 2023, doi: 10.1007/s10462-023-10577-2.

[60] Y. Ma and J. Tang, *Deep Learning on Graphs*. Cambridge University Press, 2021.

[61] F. Scarselli, M. Gori, Ah Chung Tsoi, M. Hagenbuchner, and G. Monfardini, “The Graph Neural Network Model,” *IEEE Trans Neural Netw*, vol. 20, no. 1, pp. 61–80, Jan. 2009, doi: 10.1109/TNN.2008.2005605.

[62] J. B. Lee, R. A. Rossi, S. Kim, N. K. Ahmed, and E. Koh, “Attention Models in Graphs,” *ACM Trans Knowl Discov Data*, vol. 13, no. 6, pp. 1–25, Dec. 2019, doi: 10.1145/3363574.

[63] A. G. Vrahatis, K. Lazaros, and S. Kotsiantis, “Graph Attention Networks: A Comprehensive Review of Methods and Applications,” *Future Internet*, vol. 16, no. 9, p. 318, Sep. 2024, doi: 10.3390/fi16090318.

[64] Q. Wen *et al.*, “Transformers in Time Series: A Survey,” *arXiv preprint arXiv:2202.07125*, Feb. 2022, Accessed: Aug. 20, 2023. [Online]. Available: https://arxiv.org/abs/2202.07125v5

[65] S. Li *et al.*, “Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting,” in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA: Curran Associates Inc., 2019.

[66] S. Liu *et al.*, “Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting,” in *International conference on learning representations*, 2021.

[67] H. Wu, J. Xu, J. Wang, and M. Long, “Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting,” Jun. 2021. [Online]. Available: http://arxiv.org/abs/2106.13008

[68] T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin, “FEDformer: Frequency Enhanced Decomposed Transformer for Long-term Series Forecasting,” in *International conference on machine learning*, Jan. 2022, pp. 27268--27286. [Online]. Available: http://arxiv.org/abs/2201.12740

[69] R. F. Tomlinson, “A Geographic Information System for Regional Planning,” *Journal of Geography*, vol. 78, pp. 45–48, 1969, [Online]. Available: https://api.semanticscholar.org/CorpusID:54966620

[70] R. L. Ott and M. T. Longnecker, *An Introduction to Statistical Methods and Data Analysis*, 7th ed. Cengage Learning, 2015.

[71] H. Zhou, “Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting.” Accessed: May 08, 2024. [Online]. Available: https://github.com/zhouhaoyi/Informer2020

[72] H. Zhou, J. Li, S. Zhang, S. Zhang, M. Yan, and H. Xiong, “Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting (AAAI’21 Best Paper).” Accessed: Nov. 10, 2023. [Online]. Available: https://github.com/zhouhaoyi/Informer2020

[73] X. Kong, W. Xing, X. Wei, P. Bao, J. Zhang, and W. Lu, “STGAT: Spatial-Temporal Graph Attention Networks for Traffic Flow Forecasting,” *IEEE Access*, vol. 8, pp. 134363–134372, 2020, doi: 10.1109/ACCESS.2020.3011186.

[74] D. Bahdanau, K. H. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” in *International Conference on Learning Representations, ICLR 2015,* Jan. 2015.

[75] Felix Opolka, “PyTorch implementation of the model proposed in Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting .” Accessed: Apr. 16, 2024. [Online]. Available: https://github.com/FelixOpolka/STGCN-PyTorch/tree/master?tab=readme-ov-file

[76] B. Yu, H. Yin, and Z. Zhu, “Spatio-temporal Graph Convolutional Neural Network: A Deep Learning Framework for Traffic Forecasting,” in *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, May 2017.

[77] L. Ø. Bentsen, N. D. Warakagoda, R. Stenbro, and P. Engelstad, “Spatio-temporal wind speed forecasting using graph networks and novel Transformer architectures,” *Appl Energy*, vol. 333, p. 120565, Mar. 2023, doi: 10.1016/j.apenergy.2022.120565.

[78] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, “Optuna: A Next-generation Hyperparameter Optimization Framework,” in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[79] V. Cerqueira, L. Torgo, and I. Mozetič, “Evaluating time series forecasting models: an empirical study on performance estimation methods,” *Mach Learn*, vol. 109, no. 11, pp. 1997–2028, Nov. 2020, doi: 10.1007/s10994-020-05910-7.

[80] X. Liu and W. Wang, “Deep Time Series Forecasting Models: A Comprehensive Survey,” *Mathematics*, vol. 12, no. 10, p. 1504, May 2024, doi: 10.3390/math12101504.

[81] M. Joseph, “Evaluating Forecasts – Forecast Metrics,” in *Modern Time Series Forecasting with Python*, Packt Publishing, 2022, ch. 18, pp. 469–490.

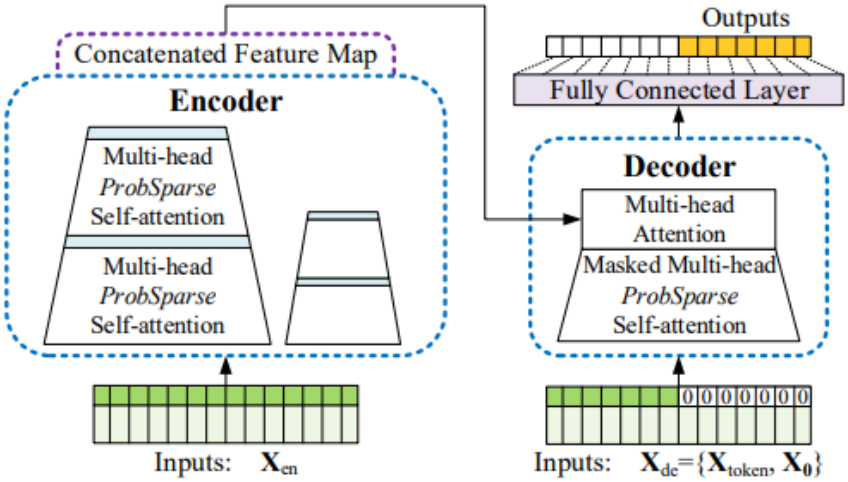
[82] A. Ross and V. Willson, *Basic and Advanced Statistical Tests*. Sense Publishers, 2017. doi: 10.1007/978-94-6351-086-8.

Appendix A

Figures

Figure A.1

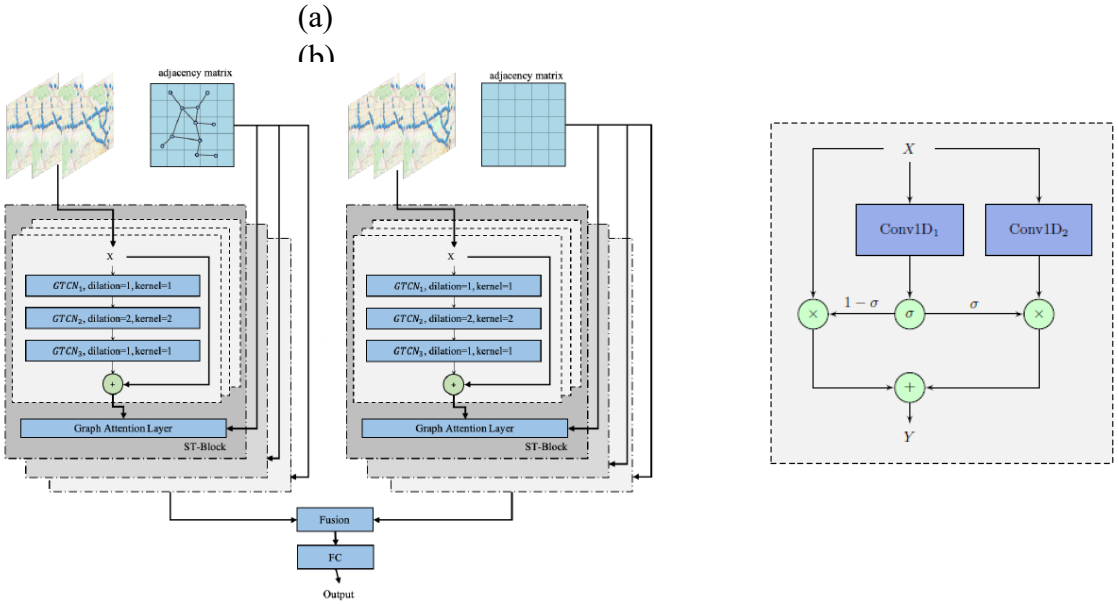
The Architecture of the Informer.



[46]

Figure A.2

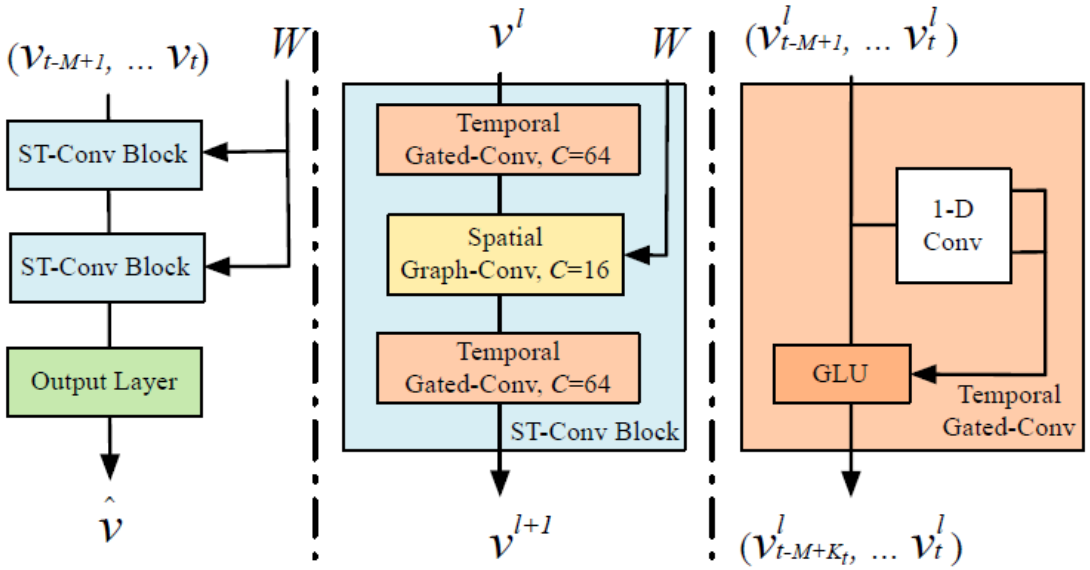
ST-GAT Architecture. (a) The Overall Architecture of the ST-GAT. (b) The Components of the GTCN Layer.



[73]

Figure A.3

The Architecture of the ST-GCN



[76]

Figure A.4

Attention Weights Visualization from a Different Batch of Data. (a) Nodes Attention Weights. (b) Edges Attention Weights.

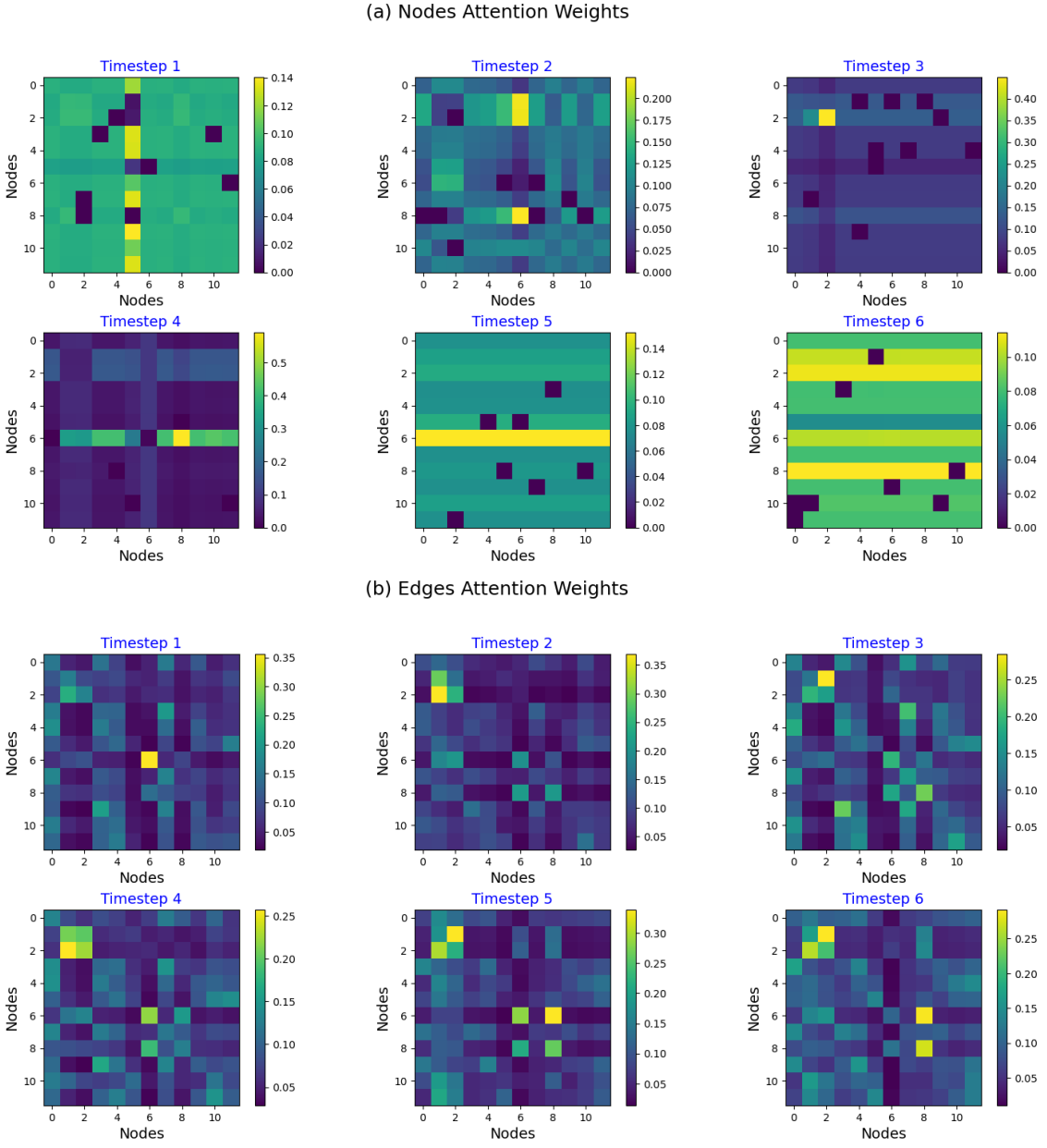


Figure A.5

Actual vs Predicted Values for Baseline Models at One Station (Beijing Dataset).

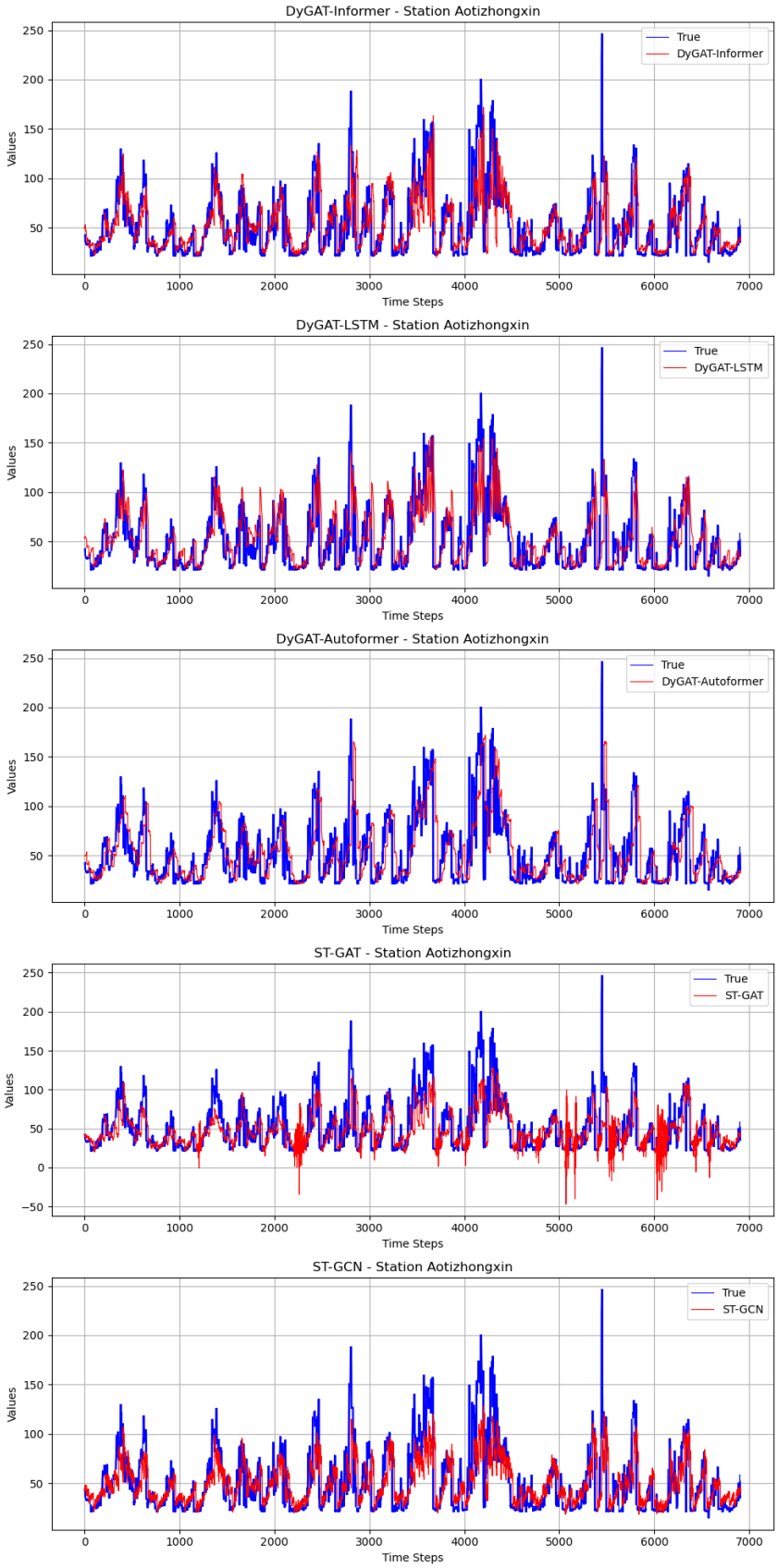


Figure A.6

Actual vs Predicted Values for Three Baseline Models (Nablus Dataset).

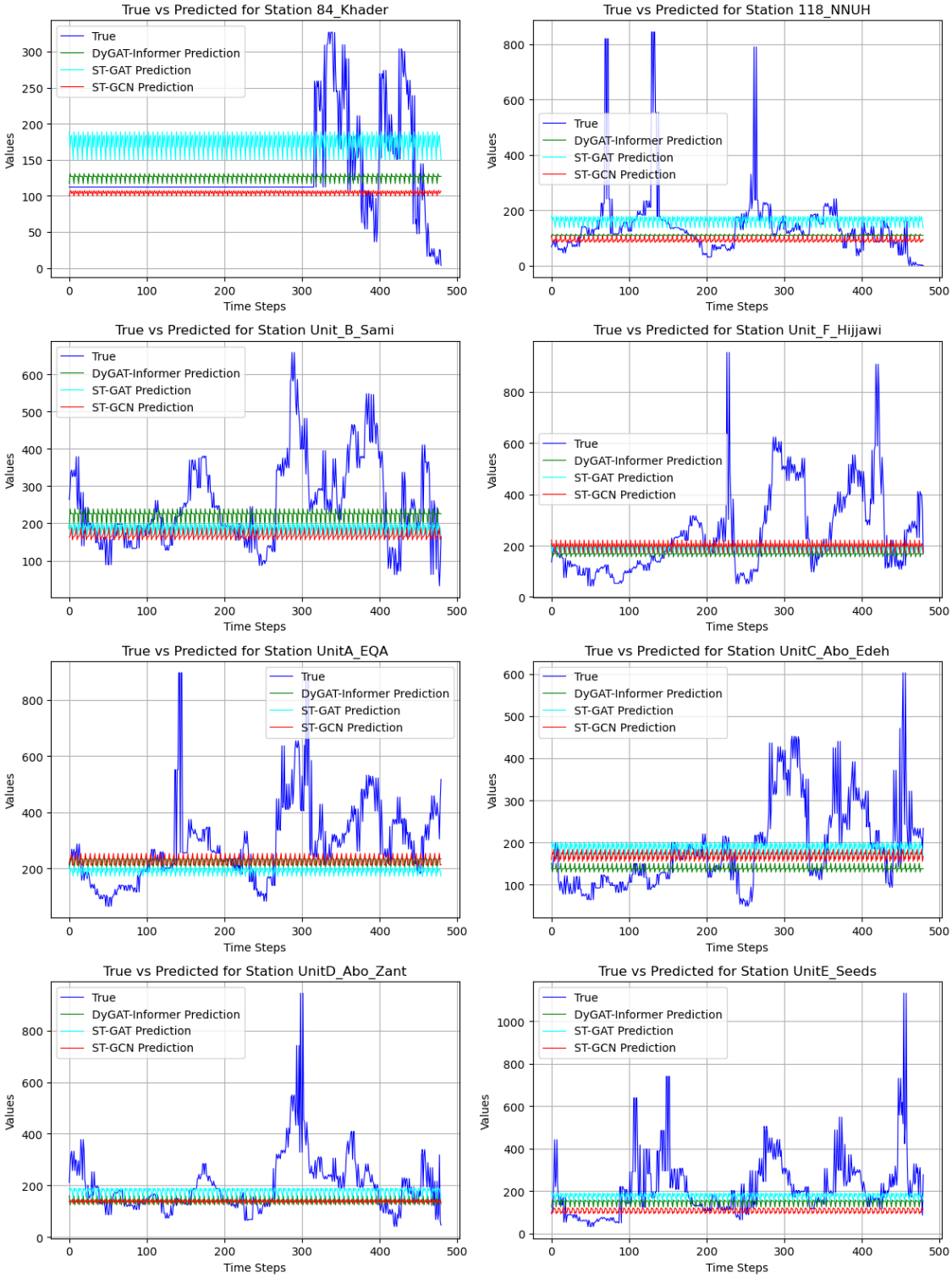


Figure A.7

Actual vs Predicted Values for DyGAT-Informer Using "Learned" Temporal Embedding (Nablus Dataset).

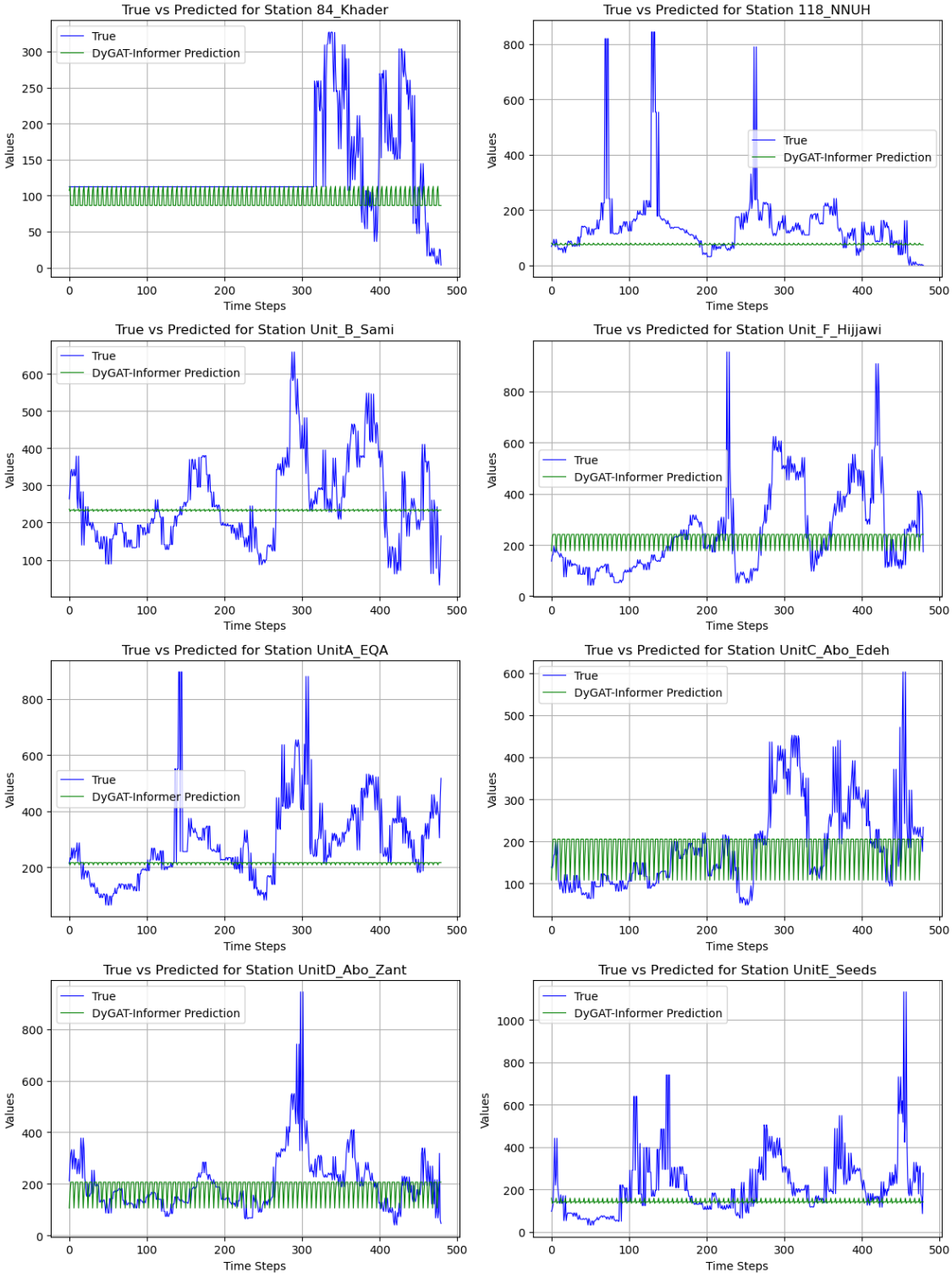
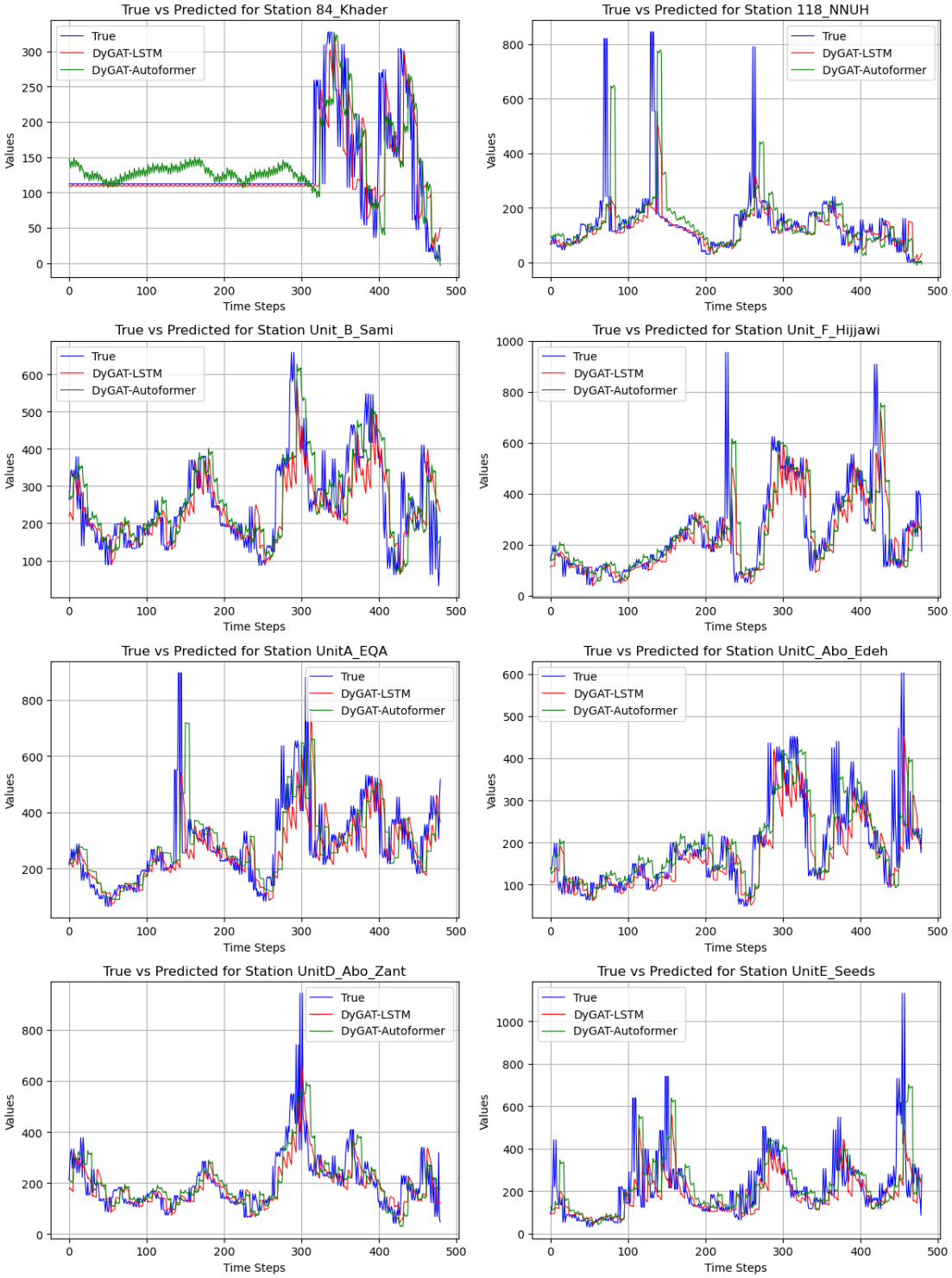


Figure A.8

Actual vs Predicted Values for DyGAT-LSTM and DyGAT-Autoformer (Nablus Dataset).



جـامعـــة النجاح الوطنية

كليـــة الدراســـات العليـــا

نموذج تعلم عميق هجين للتنبؤ بتركيزات ملوث الهواء $PM_{2.5}$

إعداد

أسماء موسى "محمد علي" مساد

إشراف

د. أنس طعمة

د. عبد الحليم خضر

قدمت هذه الرسالة استكمالا لمتطلبات الحصول على درجه الماجستير في الذكاء الاصطناعي، من كلية الدراسات العليا، في جامعة النجاح الوطنية، نابلس– فلسطين.

2024

نموذج تعلم عميق هجين للتنبؤ بتركيزات ملوث الهواء $PM_{2.5}$

اعداد

أسماء موسى "محمد علي" مساد

إشراف

د. أنس طعمة

د. عبد الحليم خضر

الملخص

يعتبر التنبؤ بجودة الهواء مجالاً بحثياً حيوياً يساعد في اتخاذ قرارات فعّالة لمكافحة تلوث الهواء. أحد أخطر الملوثات هي الجسيمات الدقيقة – ($PM_{2.5}$) ذات قطر أقل من 2.5 ميكرومتر – والتي تسبب مشاكل صحية خطيرة بسبب قدرتها على الوصول إلى الجهاز التنفسي السفلي ومجرى الدم. التنبؤ الدقيق بمستويات $PM_{2.5}$ يُعد أمراً بالغ الأهمية. على الرغم من التقدم في نماذج التنبؤ ذات الأنماط المكانية الزمانية (Spatiotemporal) المعتمدة على التعلم الآلي، إلا أن الحاجة لنماذج أكثر دقة ما زالت قائمة. يمثل استخدام النماذج العميقة الهجينة للتنبؤ بـ $PM_{2.5}$ مجالاً واعداً، حيث تهدف هذه النماذج إلى فهم العلاقات المكانية الزمانية المعقدة بشكل فعّال.

نقدم في هذا البحث نموذج DyGAT-Informer، وهو نموذج تنبؤ هجين مكاني زماني.حيث قمنا بتطوير شبكة الرسم البياني مع الانتباه الديناميكي (Dynamic Graph Attention Network (DyGAT)) لنمذجة العلاقات المكانية بين محطات القياس باستخدام خصائص الحواف المصممة بناءً على المسافة و سرعة واتجاه الرياح، مع إنشاء مصفوفة تجاور ديناميكية. ثم دمجنا DyGAT مع نموذج Informer، وهو نوع من المحولات (Transformers) المصمم للتنبؤ بالسلاسل الزمنية، مما أدى إلى بنا نموذج هجين قادر على فهم الأنماط المكانية والزمانية بشكل شامل بالإضافة إلى تحسين دقة التنبؤات

بمستويات $PM_{2.5}$. تم تقييم النموذج باستخدام بيانات من بكين تحتوي على 420,768 سجلًا على مدى أربع سنوات. تفوق نموذج DyGAT-Informer على نموذج Informer بدون مكون DyGAT وعلى نماذج أخرى استخدمت للمقارنة. حيث حققDyGAT-Informer على 50.43 في MAE، و79.9 في RMSE، و28.88% في SMAPE، مقارنة بـ 51.44 في MAE، و80.83 في RMSE، و30.25% في SMAPE وهي نتائج النموذج الذي حقق الترتيب الثالي من حيث الأداء.

كما أجرينا دراسة حالة باستخدام بيانات مدينة نابلس في فلسطين (2692 سجل لكل محطة خلال شهرين) وأضفنا معلومات جغرافية مكانية عن مصادر التلوث القريبة. نظراً لقلة السجلات، تم استبدال Informer بنموذج LSTM. حيث حقق DyGAT-LSTM المدرب مع الميزات الجغرافية المكانية الإضافية انخفاضاً بنسبة 2.08% في MAE ، و1.17% في RMSE ، و1.96% في SMAPE. رغم قصر المسافات بين المحطات، نجح DyGAT في التقاط العلاقات المكانية، حيث تفوق DyGAT-LSTM على LSTM فقط بانخفاض 3.13% في MAE، و1.48% في RMSE ، و3.67% في SMAPE، مما يؤكد فاعلية إضافة هذه البيانات وتحسين دقة التنبؤ.

الكلمات المفتاحية: تعلم عميق، نموذج تعلم عميق هجين، تنبؤ مكاني زماني، التنبؤ بالسلاسل الزمنية.