Cyprus University of Technology

杭州電子科技大學 HANGZHOU DIANZI UNIVERSITY

Master's thesis

Predictability of complex networks

Xuetong Zhao

Limassol, February 2024

MSc in Electronics Science and Technology

CYPRUS UNIVERSITY OF TECHNOLOGY

Faculty of Engineering and Technology

Department of Electrical Engineering, Computer Engineering, and Informatics

Master's thesis

Predictability of complex networks

Xuetong Zhao

Supervisor

Fragkiskos Papadopoulos

Limassol, February 2024

Approval Form

Master's thesis

Predictability of complex networks

Presented by

Xuetong Zhao

Supervisor: Fragkiskos Papadopoulos

Member of the committee: Michael Sirivianos
Member of the committee: Panagiotis Ilia

Cyprus University of Technology
Limassol, February 2024

Copyrights

Acknowledgements

First and foremost, I would like to express my sincere gratitude to all those who have supported and guided me throughout the course of my research.

I would like to thank my supervisor, Fragkiskos Papadopoulos, for their continuous guidance, valuable feedback, and unwavering support. Their expertise and insights were crucial to the success of this work. I would also like to thank Dong Zhekang for their valuable contributions and constructive discussions during the research process.

I would like to extend my appreciation to Cyprus University of Technology for providing the resources necessary for this research.

Finally, I would like to express my heartfelt thanks to my family for their unconditional love, patience, and understanding. Without their support, this work would not have been possible.

ABSTRACT

Link prediction is a key area in network science, focusing on predicting unobserved or potential future connections in a network based on the existing topology and known links. Link prediction has wide applications in fields such as social networks, recommendation systems, bioinformatics, computer networks, and financial networks. However, it faces challenges such as the lack of a unified mathematical framework and the complexity of modeling real-world network dynamics influenced by various factors. Recent advancements in network geometry have revealed that real networks can be meaningfully mapped (or embedded) into hyperbolic space. In these spaces, each node is represented by its radial (popularity) and angular (similarity) coordinates, r and θ, and nodes that are closer in hyperbolic distance are more likely to be connected. This uses the popularity-similarity model combined with the Transformer neural network to predict the radial and angular coordinate trajectories of nodes. The experimental results indicate that neural networks have stronger predictive capabilities in terms of details compared to traditional methods.

Keywords: Link Prediction, Network geometry, Time series analysis, Transformer

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

LSTM	Long Short-Term Memory
CN	Common Neighbors
AA	Adamic-Adar
RA	Resource Allocation
HDI	Hub Depressed Index
HPI	Hub Promoted Index
LHN-I	Leicht-Holme-Newman Index-I
LHN-II	Leicht-Holme-Newman Index-II
LP	Local Path
RPCA	Robust Principal Component Analysis
NMF	Non-negative Matrix Factorization
GNN	Graph Neural Networks
GCN	Graph Convolutional Networks
GAT	Graph Attention Networks
HyperGNN	Hypergraph Neural Networks
HGNN	Hypergraph Convolutional Network
PGP	Pretty Good Privacy
MLE	maximum likelihood estimation
ARIMA	AutoRegressive Integrated Moving Average
AR	autoregressive
MA	moving average
SARIMA	Seasonal ARIMA
SVM	Support Vector Machine
GRU	Gated Recurrent Units
RNNs	Recurrent Neural Networks
SGD	Stochastic Gradient Descent
Adam	Adaptive Moment Estimation
RMSprop	Root Mean Square Propagation
R^2	Coefficient of Determination
MARE	Mean Absolute Relative Error
MBD	Mean Bias Deviation
DTW	Dynamic Time Warping

1 Introduction

Link prediction is an important research in network science. It aims to predict unobserved connections (links) in a network based on its given topology and known connection information. This prediction typically relies on the information about nodes and edges in the network, using existing network data to infer future potential links or restore missing connections. The primary goal of link prediction is to uncover hidden structural relationships in the network, particularly when certain links cannot be directly observed. By inferring their existence, it provides effective predictions for network evolution. Link prediction not only enhances the understanding of the network's underlying structure but also supports network optimization and decision-making.

Link prediction has significant applications in various practical domains. In social networks, it is widely used to predict potential relationships between users, such as friend recommendations, community detection, or group division. In recommendation systems, it can suggest products, movies, or music based on users' historical behavior, thereby improving personalized recommendations. In bioinformatics, it helps discover interactions between proteins or genes, advancing disease research and drug development. In computer networks, link prediction aids in optimizing network topology, improving stability and communication efficiency to ensure efficient system operation. In financial networks, it reveals hidden market risks, identifies potential financial relationships, or tracks money flows, thus providing decision-making support.

Despite its broad applications, link prediction faces significant challenges. A major issue is the lack of a unified mathematical framework to explain the dynamics of network behavior, particularly for modeling and predicting the complex temporal evolution and structural changes in real-world networks. Specifically, there is no comprehensive theory to describe the dynamic mechanisms of node linking and unlinking. The central issue lies in determining whether network evolution is predictable and, if so, to what extent. The network evolution process is influenced by multiple factors, such as node attributes, external environments, and interaction patterns, and the specific rules governing these changes remain unclear, making it difficult to describe them using existing mathematical models.

To address this issue, it is essential to first determine whether predictable components exist within the network and analyze their regularities in depth. Once these components are identified, mathematical models can be used for effective description and prediction. Therefore, the research focus of link prediction lies in identifying the predictable components of network evolution, exploring their underlying patterns, and modeling and validating them over time. The core task is to develop models capable of capturing network evolution patterns, which may require integrating methods from graph theory, probability theory, time series analysis, and machine learning, thereby providing theoretical foundations and practical tools for link prediction.

In summary, the challenges of link prediction not only stem from the complex topology and dynamic evolution of networks but also from the difficulty in identifying and modeling the predictable components within them. Future research in this field will focus on addressing these challenges by exploring how to extract regular patterns from dynamic network evolution and employing effective mathematical tools and computational methods for modeling, thus improving the accuracy and applicability of link prediction.

The objective of this study is to embed network data with discrete topological structures into hyperbolic space to uncover the underlying evolutionary patterns of the network. Each node's spatial coordinates are represented based on its similarity and popularity [1], thereby constructing a geometric representation of the network. Hyperbolic geometry offers a novel perspective by transforming network problems into time series prediction problems. In hyperbolic space, the evolutionary trajectory of nodes can be viewed as a time series. If the evolution of node coordinates exhibits significant predictability, time series analysis can reveal the mathematical rules governing these processes, providing a unified description of network dynamics.

To further validate these predictable components, we applied time series modeling techniques combined with neural network methods, to model the evolutionary trajectories of node popularity and similarity. Neural networks can capture the long-term dependencies in node behavior, further uncovering the underlying patterns in network evolution and providing robust support for link prediction.

1.1 Main contributions

The main contributions of this research are:

(1) The trajectory analysis and prediction method using fractional Brownian motion from the paper "Fundamental dynamics of popularity-similarity trajectories in real networks," [1] this study employs neural network models to model and predict time series data. This method places greater emphasis on predicting the fluctuations and detailed patterns of the time series. Building upon the original trajectory analysis, it further investigates the predictability of these trajectories in long-term trends, resulting in predictions that are more nuanced and sensitive to temporal variations.

(2) The paper evaluates the prediction results from multiple dimensions, primarily including prediction accuracy, sequence similarity, trend matching, and detail capture. In addition to assessing prediction accuracy, the study further examines the model's ability to maintain sequence shape similarity, capture long-term trend changes, and fit local details. These multi-faceted evaluations help comprehensively reflect the model's performance in different aspects, providing a more accurate measure of its actual application effectiveness, especially in the context of complex network data prediction.

1.2 Structure of the Thesis

The second chapter reviews the main research progress in the field of link prediction and analyzes the advantages and disadvantages of various methods. The third chapter details the methodology, describing the architecture of the popularity-similarity model and Transformer neural network, as well as their application in predicting the radial and angular coordinates of nodes. The fourth chapter presents the results and discussion, comparing the experimental results with traditional method and analyzing the advantages of neural networks in detail prediction, as well as their limitations. Finally, the fifth chapter summarizes the research findings, discusses the effectiveness of the proposed method, and discusses future research directions.

Code is available at this repository:https://github.com/19857115450/Trajectory-Prediction

2 Literature Review

Link prediction is a core problem in the field of complex network science, aiming to predict connections that may exist but haven't been observed yet, as well as those that might form in the future — all based on the current structure and links within the network. Currently, the theories and methods for link prediction primarily focus on approaches such as node similarity-based methods, maximum likelihood models, dimensionality reduction techniques, and machine learning algorithms.Specifically, the first three methods are mainly designed for predicting missing links within existing networks, whereas the fourth method enables the prediction of future links.

2.1 Node similarity

Node similarity-based link prediction algorithms play a critical role in the study of link prediction in complex networks. Their core assumption is that the more similar two nodes are, the more likely they are to form a connection. Based on this assumption, many similarity metrics relying on connection mechanisms were proposed in the early stages of link prediction research. For instance, the classical Common Neighbors (CN) metric [2] posits that if two unconnected nodes share more neighbors, they are more likely to establish a connection. The formula for the CN metric is given as:

$$CN(x, y) = |N(x) \cap N(y)| \tag{2.1}$$

where $N(x)$ and $N(y)$ represent the neighbor sets of nodes x and y, and the formula counts the number of common neighbors between the two nodes.

Building upon this, the Adamic-Adar (AA) metric further adjusts for the impact of node degree on similarity calculation, reducing the weight of high-degree nodes [3]. The AA metric is represented by the formula:

$$AA(x, y) = \sum_{z \in N(x) \cap N(y)} \frac{1}{\log |N(z)|} \tag{2.2}$$

where $N(x)$ and $N(y)$ are the neighbor sets of nodes x and y, and $N(z)$ is the neighbor set of node z. The logarithmic term in the formula reduces the impact of nodes with higher degrees on the similarity calculation.

On this basis, Zhou Tao et al. introduced the Resource Allocation (RA) metric from the perspective of resource allocation [4][5][6], treating nodes as mediums for resource transmission and assigning values to nodes using the reciprocal of their degrees. The formula for the RA metric is:

$$RA(x, y) = \sum_{z \in N(x) \cap N(y)} \frac{1}{d(z)} \tag{2.3}$$

where $d(z)$ is the degree of node z, representing the resource transmission capacity of node z. Both RA and AA metrics consider the attenuation effect of neighboring nodes, but they differ in how they assign

weights to common neighbor nodes. The AA metric applies logarithmic attenuation to high-degree nodes, whereas the RA metric applies linear attenuation to the degree, reducing the impact of high-degree nodes on the similarity calculation.

Based on the CN metric, several derived similarity metrics have been introduced to account for the influence of node degrees, such as the Hub Depressed Index (HDI) [5], Hub Promoted Index (HPI) [6], Salton metric [7], Leicht-Holme-Newman Index-I (LHN-I) [8], and Jaccard metric [9]. These methods primarily rely on information between two nodes to construct similarity metrics, but they do not fully utilize the structural information of the entire network.

As the complexity of networks increases, methods that rely solely on information between pairs of nodes have revealed certain limitations. To address these limitations, some researchers have proposed similarity metrics based on network paths, such as Leicht-Holme-Newman Index-II (LHN-II) [8], Local Path (LP) metric [10], and Katz metric [11]. These methods consider the similarity between nodes based on different paths between them, expanding the application scope of similarity metrics. However, similarity-based methods still face certain limitations. They tend to strongly depend on the structural features of the network, resulting in significant performance differences under different network structures. Furthermore, these methods have less applicability when partial common neighbor information is missing. Therefore, future research needs to explore how to incorporate more global network information to enhance the robustness and applicability of similarity-based methods.

It is important to note that these methods are inherently dependent on the static snapshot of the current network, which limits their applicability to future link prediction scenarios.

2.2 Structural likelihood probability

Link prediction algorithms based on structural likelihood probability are an important method in complex network analysis. These algorithms rely on mathematical modeling to explain and predict potential connections between nodes in the network. Typically, these algorithms involve three main steps. The first step is to construct a likelihood probability model of the network structure, with the core of the model being to derive the probability of connections forming between nodes based on the existing network structure. Next, the algorithm maximizes the likelihood probability of the observed network structure, which typically involves optimization based on predefined network organizational principles or connection mechanisms. For example, in some cases, the hierarchical structure, community structure, or other geometric properties of the network may influence the connection probability between nodes.

In 2008, Clauset et al. proposed a link prediction method based on a hierarchical structure likelihood model. This method reveals the generation mechanism of complex networks by analyzing the hierarchical relationships between nodes in the network. The model assumes that network connections are not formed randomly, but are constrained by higher-level structures. Specifically, Clauset et al. established a probability model for hierarchical structures to calculate the likelihood probability of connections between nodes and used this model for link prediction. This hierarchical structure likelihood model can uncover the potential connectivity of the network by analyzing the relative position and adjacency relationships between nodes, especially in hierarchical network structures such as social networks or citation networks.

As research in this field has advanced, many scholars have proposed different link prediction methods based on network structural likelihood probability. For instance, Pan et al. [12] proposed a method to quantify the likelihood of a connection by analyzing the impact of link addition and removal on the overall likelihood probability of the network. They used a dynamically updated network model to calculate the connection probability of each node pair and predicted future connections by comparing the likelihood probabilities of different connection patterns. Another example is the likelihood model based on random block modeling proposed by Gaucher et al. [13], which is mainly applied to link prediction in sparse networks. The random block model fits the network structure by constructing a network with multiple communities and uses this model to identify missing links in the network, especially in cases of incomplete information or sparse networks.

Although link prediction methods based on network structural likelihood probability have rigorous mathematical models and can effectively establish the relationship between the macrostructure of the network and micro-level links, these methods also have some drawbacks. First, they often rely on specific network organizational principles, such as hierarchical structures, community structures, or locality principles. As a result, these methods may not be suitable for handling all types of network structures, particularly in networks with complex or highly dynamic structures. Additionally, while network structural likelihood probability provides an effective predictive framework, its computation and reasoning processes are typically complex [14]. The complexity and computational load increase significantly, especially when dealing with multi-level or multi-dimensional networks.

To address these issues, researchers have started to explore how to maximize the likelihood probability model of the network structure within a broader and more flexible network organizational framework. In recent years, new methods have proposed more general network structure models that can adapt to more complex and irregular network types, further expanding the application of these methods. In addition to improving the universality of the models, another key research direction is how to combine the structural likelihood probability information with other efficient link prediction algorithms. By incorporating emerging technologies such as machine learning and graph neural networks, future research is expected to propose more efficient and accurate link prediction methods, which will not only handle complex network structures but also improve prediction accuracy and computational efficiency.

Link prediction methods based on network structural likelihood probability have significant potential in practical applications, especially in fields like social networks, communication networks, and recommendation systems. Through further research and optimization of these methods, link prediction and network analysis in the future will be more accurate and efficient. These advancements will help address various challenges in real-world networks, such as incomplete node information, network topology changes, and large-scale data processing issues.

Likewise, as these methods are fully dependent on the present network structure, they are primarily applicable to the identification of missing links, rather than the prediction of future connections that may arise as the network evolves.

2.3 Dimensionality reduction-based link prediction algorithms

Dimensionality reduction-based link prediction algorithms have become an important solution for link prediction in complex networks due to their strong mathematical theoretical background and excellent algorithmic interpretability. As the scale of data and the complexity of network structures increase, traditional similarity-based link prediction methods gradually reveal limitations, especially in dealing with large-scale sparse networks. Dimensionality reduction methods map high-dimensional network data to low-dimensional space, not only improving the efficiency of the algorithm but also revealing deeper structural features. The core advantage of these methods lies in their excellent mathematical interpretability and strong theoretical foundation, which makes dimensionality reduction methods highly promising for link prediction in large-scale complex networks. These methods are mainly divided into two categories: matrix factorization and network embedding.

In matrix factorization methods, the sparse linear representation of the network's adjacency matrix is a common technique. By performing sparse linear representation, noise connections in the network can be effectively removed, and the network topology can be reconstructed, making it easier to identify missing or false connections. The Robust Principal Component Analysis (RPCA) method proposed by Candès et al. [15] decomposes high-dimensional data matrices into low-rank matrices and error matrices, with the goal of minimizing the norm of the error matrix to recover the principal components of the high-dimensional data matrix. In link prediction, this method improves the interpretability of the data and the accuracy of the prediction by removing noise. Liu et al[16]. proposed an improved low-rank matrix decomposition method suitable for cases where matrix elements are distributed across different subspaces. This method retains more information and better handles the sparsity of networks. Additionally, some studies have introduced multi-hypergraph interactions to characterize higher-order manifold information during the sparse linear representation of matrices [17], further improving the accuracy of matrix representations. By using a linear combination of sparse matrices and the observed network adjacency matrices, some studies have reconstructed the network topology and proposed optimization methods[18]. For example, Xian et al. [19] optimized sparse matrices and performed network topology reconstruction through a linear combination with the observed adjacency matrix, enhancing the link prediction performance.

Another matrix factorization method is Non-negative Matrix Factorization (NMF), which has been widely applied in link prediction problems. The core idea of NMF is to map complex networks into a latent feature space, thereby revealing deeper hidden structural information within the network. NMF decomposes the adjacency matrix of the network into the product of two non-negative matrices, effectively capturing potential patterns in the network, especially in uncovering the hidden connections between nodes. Classic link prediction algorithms based on NMF include FSSDNMF and RGNMF-AN [20], as well as link prediction algorithms in temporal networks [21]. These algorithms use matrix factorization techniques to reveal latent patterns in the network structure and predict missing links. NMF has also been successfully applied to link prediction in temporal networks [21], where it effectively captures the evolving relationships between nodes over time.

For network embedding learning methods, inspired by machine learning on structured data, these approaches sample from complex networks to learn low-dimensional representation vectors of nodes, and

have achieved promising results in link prediction [22] [23] [24].Existing network embedding methods can generally be categorized into Euclidean embedding methods and non-Euclidean embedding methods.

Euclidean embedding methods are exemplified by DeepWalk, a pioneering approach that introduced ideas from natural language processing into network representation learning [25]. By sampling nodes in a network to generate ”node sequences” (analogous to sentences), the algorithm embeds network nodes into a low-dimensional vector space based on the word2vec approach from NLP [26], thereby transforming link prediction into a problem of measuring similarity between node embeddings.Other notable network embedding methods include Node2vec [27], LINE [28], and CNDE [29]. Link prediction algorithms based on network embedding tend to have high time complexity, and their prediction accuracy varies across different types of networks.While such methods have demonstrated strong performance across various tasks, they often struggle to effectively represent networks with complex hierarchical structures or power-law degree distributions, and typically require high-dimensional embeddings to capture such patterns.

To better capture the latent geometric properties of these networks, non-Euclidean embedding methods, particularly those based on hyperbolic space, have been proposed and extensively studied. Hyperbolic spaces have negative curvature and expand exponentially, making them well suited for modeling the hierarchical and scale-free nature commonly observed in complex networks. In these methods, nodes are embedded into manifolds of hyperbolic space, and the geometric distance between nodes serves as a proxy for connection probability, often modeled using functions such as the Fermi-Dirac distribution or hyperbolic distance metrics. Representative approaches include Poincaré embedding [30], the Lorentz embedding [31], and Mercator embedding [32] [33] [34]. These methods not only preserve network structure in low-dimensional spaces but also demonstrate superior performance in link prediction tasks, and are especially effective for networks with hierarchical, sparse, or temporally evolving structures.

Similar to the previous two categories of methods, dimensionality reduction-based link prediction approaches are still primarily designed for the completion of missing links within the existing network structure. Although some methods have shown potential in capturing network evolution patterns, they are not inherently capable of directly predicting future changes in connectivity.

2.4 Machine learning

With the explosive development of machine learning, link prediction in complex networks has also entered a new wave of progress. Many outstanding research achievements in link prediction are closely related to machine learning. Among these, graph neural networks are one of the technologies most closely connected to link prediction in complex networks. Since graph neural networks can effectively capture structural information and organically integrate node attribute information, their application to link prediction in complex networks is gaining increasing attention from researchers. Graph neural networks allow direct training and updating of node representation vectors on network (graph) structures [35] [36] [37] [38]. This significant breakthrough has made graph neural networks a hot research topic, distinct from traditional neural networks. Furthermore, it has been proven that graph neural networks show strong potential in link prediction tasks for complex networks [39] [40] [41].

Graph neural networks are a class of machine learning methods that operate directly on unstructured data

with graph (complex network) structures [35] [36] [37]. Welling et al. [35] first proposed the graph convolutional neural network based on spectral graph theory, which updates node features by integrating information from neighboring nodes. The GraphSAGE algorithm [36] uses sampling and aggregation of neighbor node features to train a learnable function for generating node representation vectors, enabling an inductive learning paradigm for graph learning. Going a step further, Veličković et al. [37] proposed the graph attention network (GAT), which automatically learns how to combine the features of neighboring nodes. Notably, GAT can belong to either the transductive or inductive learning paradigm. However, ordinary graph neural networks lack the capability to capture high-order structural information inherent in complex networks, and stacking too many layers can lead to over-smoothing [42].

To better capture high-order information in networks, researchers have proposed hypergraph neural networks [43]. Hypergraph neural networks are a class of machine learning methods that can directly learn high-order structural information present in hypergraph structures. In the early stages of hypergraph neural network development, some scholars were inspired by graph neural networks and proposed the HyperGCN framework [37], which trains graph neural networks on hypergraph structures using semi-supervised learning. In [44], Feng et al. first proposed the hypergraph convolutional neural network (HGNN) framework based on hypergraph spectral theory, enabling the encoding and learning of high-order data relationships in hypergraph structures. Further, Feng et al. [45] proposed a more general model framework, HGNN+, for learning high-order structural information based on HGNN. Dong et al.[46] introduced HNHN, a hypergraph convolutional neural network where both hypernodes and hyperedges have nonlinear activation functions, allowing for flexible adjustment of the importance of hypernodes and hyperedges according to dataset characteristics. Bai et al. [47] enhanced the representational learning ability of hypergraph convolution by adding an end-to-end attention mechanism on top of hypergraph convolution. Since graph neural networks can effectively learn the spatial features of network (graph) structures, they are widely used in link prediction tasks for complex networks [39] [40] [41]. Compared to various types of graph neural networks, hypergraph neural networks can learn high-order structural information within complex network structures, indicating that they are also more suitable for link prediction tasks in complex networks [48].

In addition to the four common categories of link prediction algorithms mentioned above, some researchers have creatively proposed many other excellent link prediction methods based on different theoretical foundations. For example, Zhang et al. [49] proposed the S-AIC and S-BIC link prediction models based on model averaging methods; Singh et al. [50] introduced the FLP-ID link prediction algorithm grounded in fuzzy set theory; and Lai et al. [51] proposed the BMPA algorithm based on modularized belief propagation. These research efforts, rooted in diverse theoretical frameworks, have significantly expanded the boundaries of link prediction in complex networks. They have also helped establish effective bridges between link prediction and various academic disciplines, demonstrating the interdisciplinary strengths of complexity science.

Although the aforementioned GNN-based and hypergraph-based link prediction methods have achieved notable progress in capturing structural patterns within complex networks, they are fundamentally constrained to static graph settings. These models typically operate on a fixed snapshot of the network and assume that the topology remains unchanged during the learning and inference processes. However, real-world networks are inherently dynamic, with nodes and edges continuously appearing, disappearing, or

evolving over time.

To address the aforementioned limitations of static graph models, a growing body of research has focused on developing Dynamic Graph Neural Networks (DGNNs), which explicitly incorporate temporal information into the graph learning process. These models are designed to capture both the evolving topological structure and the temporal dependencies that arise in dynamic or time-evolving networks. By modeling changes in node interactions over time, DGNNs enable the learning of time-aware node representations that reflect both current and historical states of the network. As a result, these methods are capable not only of identifying missing links within the current graph snapshot but also of forecasting future links that may form as the network continues to evolve.

Representative DGNN models include Temporal Graph Networks (TGN) [52], which combine a memory module with a message-passing architecture to continuously update node states in response to temporal interaction events, enabling fine-grained temporal reasoning in event-driven networks. DyRep [53] introduces a framework based on temporal point processes to jointly model both node dynamics and edge dynamics over time. EvolveGCN [54], in contrast, takes a model-level perspective, using a recurrent neural network to update the parameters of a GCN over time, thereby allowing the graph convolution operation itself to adapt to evolving network structures.

Although dynamic graph neural networks (DGNNs) have demonstrated strong capabilities in modeling evolving network structures and capturing temporal dependencies, most existing studies remain focused on short-term link prediction tasks. These models are typically trained to predict the next few interactions or structural changes in the near future, often within a limited number of time steps or a narrow temporal window. This focus is largely driven by the availability of training data, the relative stability of short-term patterns, and the ease of evaluation. However, in many real-world applications—such as recommendation systems, social interaction modeling, and scientific collaboration forecasting—the ability to predict long-term network evolution is of critical importance.

Long-term future link prediction, which involves forecasting connections that may form over a significantly extended time horizon, remains an underexplored and challenging area. As the prediction horizon increases, structural uncertainty grows, new nodes and patterns may emerge, and existing temporal dependencies may weaken or shift. Moreover, the lack of sufficient supervision signals and the tendency of errors to accumulate over time further hinder the effectiveness of traditional DGNN-based models in long-range scenarios. Therefore, despite the theoretical potential of DGNNs to address this task, there is still a clear gap in the current research landscape when it comes to robust, generalizable, and scalable solutions for long-term link prediction.

However, when studying link prediction solely from the perspective of algorithm design, it is difficult to answer a key question: ”What is the upper limit of link prediction accuracy for a specific complex network?” In fact, even for the same complex network, different link prediction algorithms often exhibit varying levels of prediction accuracy. This indicates that the accuracy of a particular algorithm only reflects its ability to interpret the generative mechanism of the specific network structure, but does not fully capture the inherent predictability of the network’s links. Thus, quantifying and measuring the link predictability of complex networks has become a critical research question [37]. For example, when the prediction results of a link prediction algorithm are unsatisfactory on a certain complex network, we cannot determine whether the algorithm is poorly chosen or if the network’s links themselves are

inherently difficult to predict accurately. If we can determine the degree of ”predictability” of a network’s links, we will be able to effectively address this issue. To some extent, the link predictability of a network can be regarded as an important intrinsic property of the network itself.

At present, some scholars have focused on and studied the link predictability of complex networks . Yin et al. [55] studied the impact of Shannon entropy on link predictability by combining local information of networks with Shannon entropy. Lü et al. employed matrix perturbation theory to reconstruct the link structure of networks and proposed the concept of ”network structure consistency,” arguing that ”link predictability” is an important inherent property of networks. Based on this, Chen et al. [56] combined network feature statistics with network structure consistency and proposed a link predictability metric based on degree clustering coefficients. Chen et al. [57] studied the link predictability of bipartite networks using non-parametric structural enhancement and structural perturbation methods. Sun et al. [58] discovered, through research on 12 real-world networks, that the normalized shortest compression length can directly evaluate link predictability and shows a clear linear relationship with it. Tan et al. [59] analyzed network topology from the perspective of eigen-spectra and proposed a mathematical metric for measuring the link predictability of complex networks based on eigen-spectral theory.

These research achievements have significantly advanced the study of link predictability in complex networks. Delving deeply into the link predictability of complex networks not only provides critical guidance for the design of link prediction algorithms but also enhances our understanding of network structure generation mechanisms. Currently, although some progress has been made in the research on link predictability of complex networks, the field is still in its infancy. Significant challenges remain in establishing a mathematical upper bound for link prediction accuracy or proposing universal and effective link predictability metrics.

3 Research Methodology

Modeling and prediction of network dynamics are not just aimed at understanding the current structure of the network, but also at predicting potential changes that may occur at a future point in time, particularly those changes that affect the network's performance, efficiency, reliability, and stability. Through modeling and prediction, we can more effectively manage networks, prevent potential problems, and make targeted decisions.

Significant progress has been made in the mathematical modeling and methodologies of time series data in various complex dynamical systems, particularly in areas such as gravitational systems, fluid dynamics, molecular dynamics, and financial market data. The modeling of these systems has not only advanced the theoretical development of the related disciplines but also provided strong support for practical applications. In recent years, the development of mathematical methods and models has achieved considerable maturity, especially over the past few decades, where researchers have made substantial progress in describing the dynamic behavior of these systems using mathematical tools.

However, in-depth research into controlling network dynamics and the associated mathematics still faces many challenges. These challenges primarily arise from the complex interconnections and topological structures between nodes in a network. Specifically, the dynamic behavior of networks is closely related to their structure, and the connectivity of network nodes often changes over time, typically exhibiting non-linear characteristics. This differs from traditional classical dynamical system models, which in many cases rely on more fixed structures and time series data patterns.

Therefore, the modeling of complex network dynamics and its related fields still require in-depth research into the non-linearity and dynamic changes of network topologies. This not only demands more refined modeling and computation from a mathematical modeling perspective but also requires adaptive prediction and control during the actual evolution of the network, particularly when dealing with large-scale, dynamically changing networks. Although some progress has been made, how to understand and effectively influence network dynamics remains an open research question that warrants further exploration in both academic and practical fields.

For link prediction in complex networks, most methods typically rely on directly analyzing the network's topological structure. However, traditional topological network analysis methods often face challenges in practical applications. This is because complex networks are composed of discrete nodes and edges, forming a topological structure, while many dynamic systems have standard time series data. The discreteness of this topological structure makes link prediction more difficult to analyze.

In recent years, the development of hyperbolic geometry has provided a new perspective to address this issue. Research has shown that many real-world complex networks have an inherent geometric structure that is not easily captured by Euclidean space. To better understand and predict the dynamic behavior of complex networks, scholars have proposed a method of embedding the network into hyperbolic space. Hyperbolic space is a type of geometry with negative curvature, which can effectively represent the hierarchical structures and relationships between nodes in a network, features that are often present in many real-world networks.

Once a complex network is embedded into hyperbolic geometry space, its structural features are trans-

formed into continuous geometric representations, making the network's dynamic evolution similar to traditional time series prediction tasks. This mapping converts the nodes and edges in the network into points and paths in hyperbolic space, where the relative positions of nodes not only reflect their topological relationships but also capture characteristics such as node popularity, similarity, and other potential structural features.

In this hyperbolic geometry space, the evolution of the network no longer solely relies on traditional discrete topological reasoning. Instead, it captures the network's evolution patterns by observing the changes in node trajectories. These trajectories represent the movement of nodes over time, reflecting the dynamic changes in the connections between nodes. Therefore, the evolution trajectories of nodes can be regarded as time series data, similar to time series prediction tasks encountered in other fields.

By embedding the network into hyperbolic geometry space, existing time series modeling methods can be employed to predict the future evolution of the network. These methods are capable of capturing the complex relationships between nodes and the evolution patterns of the topological structure, providing powerful tools for link prediction in dynamic networks. Specifically, time series modeling methods predict future trajectories by learning from historical node trajectory data.

As shown in Figure 3.1, this is the overall flowchart . This flowchart illustrates the entire process from data preparation to experimental analysis. First, the data preparation phase involves obtaining the original edge connection data. Then, in the data processing phase, hyperbolic geometric embedding and similarity popularity trajectory extraction are performed. In the time series construction and data analysis stages, the intrinsic patterns and distribution characteristics of the data are analyzed. Next, an appropriate modeling method is chosen for predictive modeling, followed by model training and parameter adjustment. Finally, in the model evaluation stage, evaluation metrics are used to assess the model's performance, and the predicted results are compared with the actual results. The process concludes with the analysis of experimental results to draw conclusions.

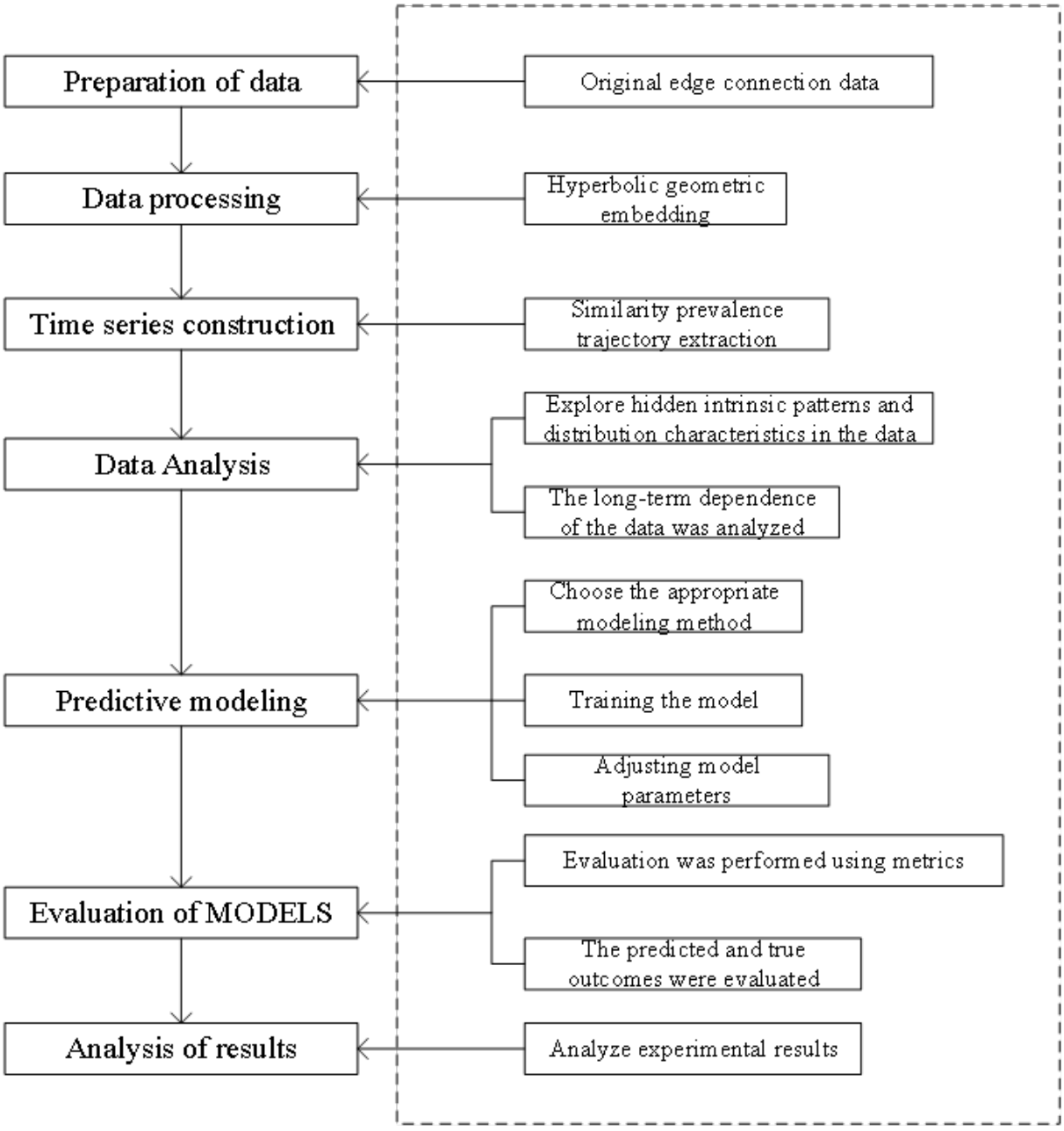

Figure 3.1: Overall flowchart.

3.1 Data Preparation and hyperbolic geometric embedding

3.1.1 Dataset

In this study, three real-world network datasets are used, all sourced from the work cited in Reference [1].

(1) The flight connections between airports in the United States [60] are based on the national aviation network data.The dataset originates from the publicly available flight information provided by the U.S. Department of Transportation, covering flights from January 1988 to May 2015. Based on daily flight connection data during this period, 10,000 network snapshots were created, with each

snapshot representing a single day's flight network. In these snapshots, nodes represent airports, and edges denote flight connections between them. To simplify the network structure, bidirectional flight relationships were converted into undirected edges, and the largest connected subgraph was extracted from each snapshot [1]. Figure 3.2 shows a schematic of the US Air transportation network.

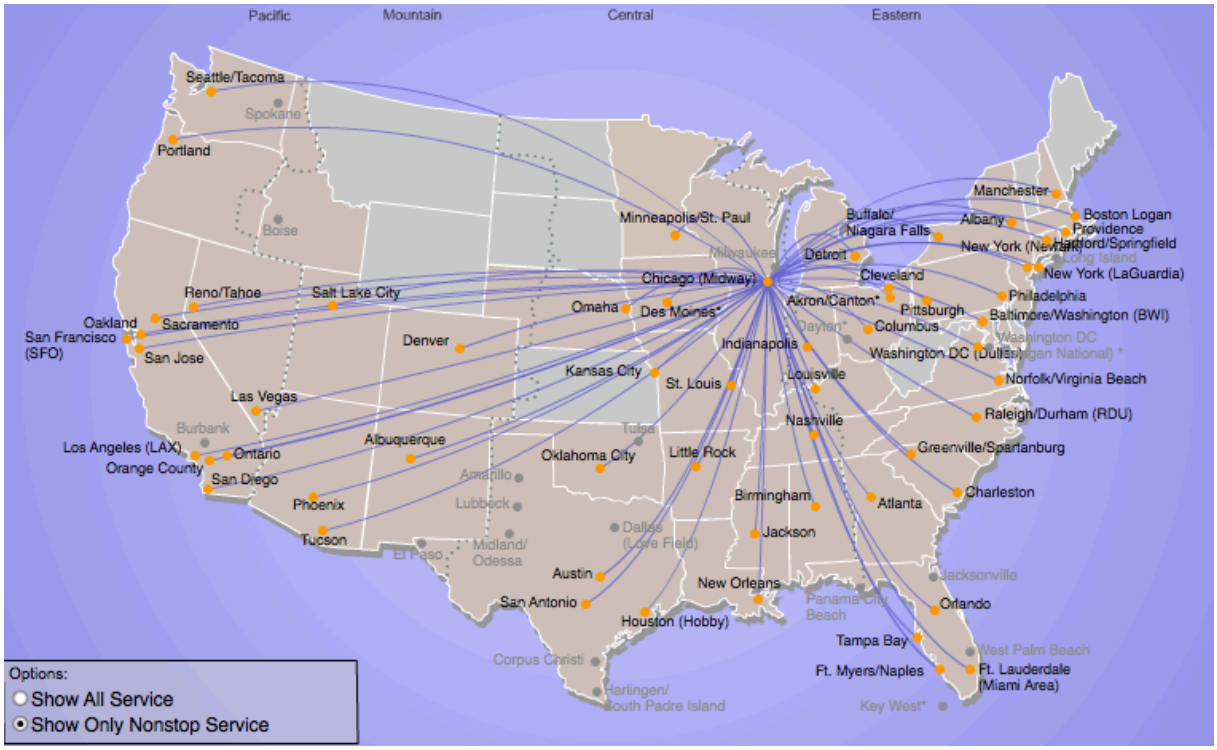

Figure 3.2: Airport network connectivity diagram.

(2) The arXiv collaboration network [61] dataset originates from the arXiv platform and is used to analyze author collaboration relationships in academic papers. Each paper is classified into different fields, and authors establish collaborations by co-authoring papers. In this network, nodes represent authors, and edges represent their collaboration relationships, i.e., two authors co-authored a paper. The dataset shows the dynamic changes in author collaboration, with new collaborations forming over time while some old collaborations may fade. By analyzing these collaboration data, key researchers, research trends, and interdisciplinary collaboration patterns within academic fields can be identified [1]. Figure 3.3 illustrates the schematic of the arXiv collaboration network.

Figure 3.3: The arXiv collaboration network connectivity diagram.

(3) This dataset originates from the PGP (Pretty Good Privacy) Web of Trust (WoT) [62], a decentralized trust model based on public-key encryption. In PGP, users verify each other's public keys by digitally signing them, thereby establishing trust relationships. The nodes in the graph represent PGP users and their corresponding public keys, with each node containing the user's identity information and public key. The edges represent trust relationships between nodes. In this trust network, trust is propagated through endorsements between nodes. When a user digitally signs another user's public key, it means they are validating that user's identity, creating a trust chain within the network. This dataset, collected by Jörgen Cederlöf and others, includes trust relationships and public key information between PGP users, reflecting the distribution characteristics of nodes and edges in the trust network. It is widely used in research on trust propagation, network topology, and security issues in encrypted communication [1]. Figure 3.4 illustrates the schematic of the PGP Web of Trust connectivity.

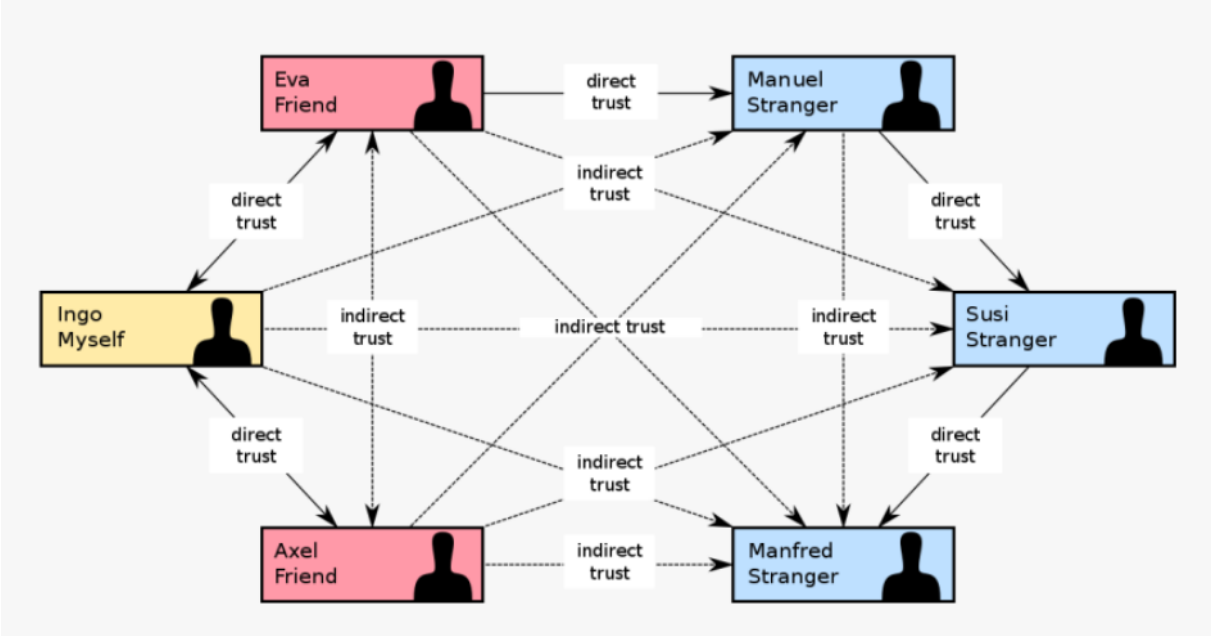

Figure 3.4: PGP Web of Trust connectivity diagram.

3.1.2 Hyperbolic geometric embedding

Representative hyperbolic embedding methods include Poincaré embedding [30], Lorentz embedding [31], and Mercator embedding [34].

The Poincaré model [30]is a commonly used representation in hyperbolic geometry, typically using a unit disk (in two dimensions) or a unit sphere (in higher dimensions) to represent hyperbolic space. The core of this model is the use of the Poincaré distance function to compute the distance between two points in space, defined as:

$$d_\rho(x, y) = \text{arcosh}\left(1 + \frac{2\|x - y\|^2}{(1 - \|x\|^2)(1 - \|y\|^2)}\right) \tag{3.1}$$

where x and y are points within the unit disk, and $\|x\|^2$ and $\|y\|^2$ represent the distances of points x and y from the origin. The Poincaré model is highly intuitive and suitable for visualizing hierarchical structures, such as taxonomies and social networks. However, it may encounter numerical instability issues in high-dimensional spaces, and its optimization can be trapped in local minima, making it challenging to achieve effective global optimization, especially in low dimensions.

The Lorentz model [31], on the other hand, is a representation of hyperbolic geometry well-suited for efficient Riemannian optimization. It represents points in space through the upper half of a high-dimensional hyperboloid, with coordinates $x = (x_0, x_1, \ldots, x_n)$ satisfying $x_0^2 - \|\mathbf{x}\|^2 = 1$, where $\mathbf{x}$ refers to all coordinates except x_0. The Lorentz model's distance function is given by:

$$d_L(x, y) = \text{arcosh}(-\langle x, y\rangle_L) \tag{3.2}$$

where $\langle x, y\rangle_L = -x_0 y_0 + \sum_{i=1}^{n} x_i y_i$ is the Lorentzian scalar product. Compared to the Poincaré model, the Lorentz model offers stronger stability and optimization efficiency when handling high-dimensional

data, especially in large-scale datasets. During Riemannian optimization, the Lorentz model avoids the numerical instability issues that arise in the Poincaré model, making it more efficient in the optimization process. While its geometric structure is less intuitive than that of the Poincaré model, the Lorentz model excels in high-dimensional spaces and complex datasets, particularly in tasks that require efficient learning of hierarchical structures.

In this study, we use trajectory data obtained through Mercator embedding, all of which is sourced from Reference [1].

The Mercator method was employed to map these network snapshots into the underlying hyperbolic space [34]. At its core, the Mercator method combines Laplace eigenmaps with maximum likelihood estimation (MLE) to produce precise and efficient network embeddings. This method has been applied in previous studies and uses MLE to optimize the positions of nodes in hyperbolic space.

In a nutshell, Mercator takes as input the network's adjacency matrix A. The generic element of the matrix is $A_{ij} = A_{ji} = 1$ if there is a link between nodes i and j, and $A_{ij} = A_{ji} = 0$ otherwise. It then infers radial (popularity) and angular (similarity) coordinates, r_i and θ_i, for all nodes $i \leq N$. To this end, it maximizes the likelihood function:

$$L = \prod_{1 \leq i,j \leq N} p(x_{ij})^{A_{ij}} \left[1 - p(x_{ij})\right]^{1 - A_{ij}} \tag{3.3}$$

where the product goes over all node pairs i, j in the network, while $p(x_{ij})$ is the Fermi-Dirac connection probability:

$$p(x_{ij}) = \frac{1}{1 + e^{x_{ij} - R}} \tag{3.4}$$

where $x_{ij} = r_i + r_j + 2 \ln\left(\frac{\Delta\theta_{ij}}{2}\right)$ is approximately the hyperbolic distance between nodes i and j; R is the radius of the hyperbolic disk where nodes reside; and $T \in (0, 1)$ is the network temperature, which is also inferred by Mercator and is related to the clustering strength of the network.

The connections and disconnections among nodes act respectively as attractive and repulsive forces. Mercator feels these attractive/repulsive forces, placing connected (disconnected) nodes closer to (farther from) each other in the hyperbolic space. We note that changes in the adjacency matrix result in the re-evaluation of all node positions.

The Figure 3.5 shows the visualization result of applying Mercator embedding to any given topology snapshot in the US airport network.

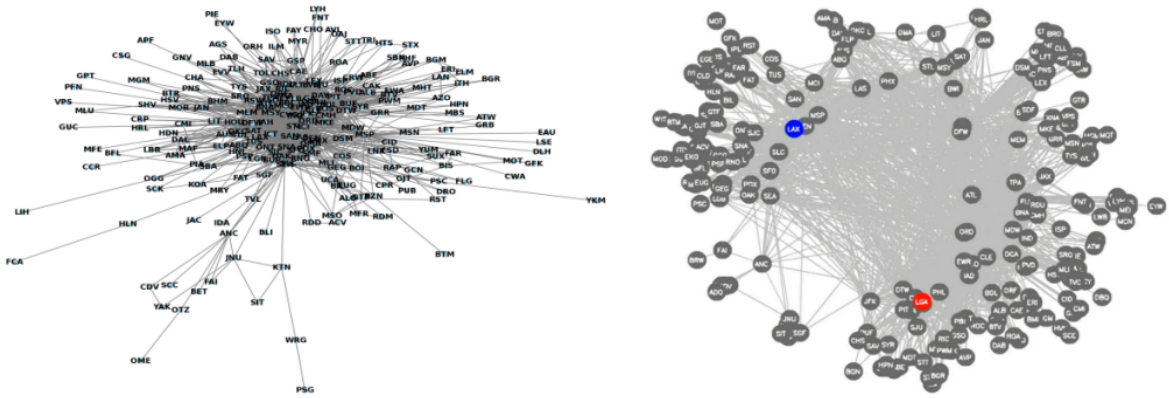

Figure 3.5: Original network(left) and the network after Mercator embedding(right)[1].

This approach effectively maps the complex topological structure of the network into hyperbolic space, capturing the latent relationships between nodes. The Mercator method handles network sparsity effectively, preserves structural integrity, and significantly reduces computational complexity. Moreover, the node coordinates generated by the Mercator embedding can be used for subsequent link prediction tasks, such as predicting potential future connections by analyzing the trajectories of node similarity and popularity.

The Mercator method excels in processing large-scale sparse networks, producing embeddings that reflect high-order relationships between network nodes with scalability and accuracy. Through this hyperbolic embedding approach, the underlying structural features of the network are effectively captured, providing a powerful tool for analyzing network dynamics and performing link prediction.

3.2 Time series analysis

By using the Mercator projection method to process topological networks, the complex link prediction problem transformed into a time series prediction problem, significantly simplifying the prediction and analysis. Taking node BIL(Billings Logan International Airport) from the USAir dataset as an example, the trajectory maps extracted using the Mercator method are shown in Figure 3.6 to Figure 3.8, corresponding to the popularity trajectory, similarity trajectory, and expected degree trajectory, respectively. (The data used in this experiment comes from the trajectory data extracted after Mercator embedding, as published in [1]):

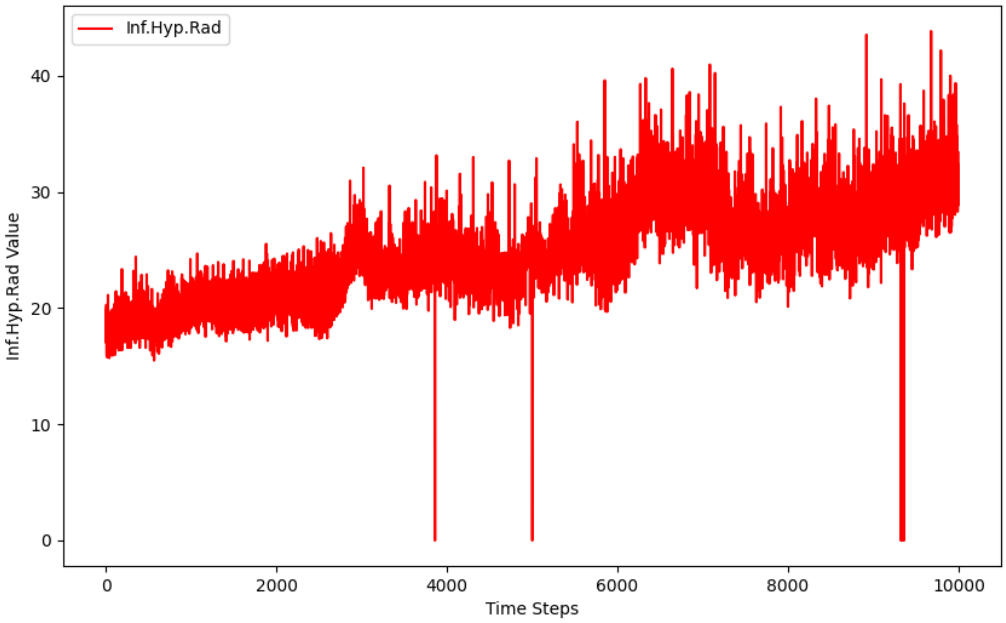

Figure 3.6: In USAir, the node BIL, Popularity

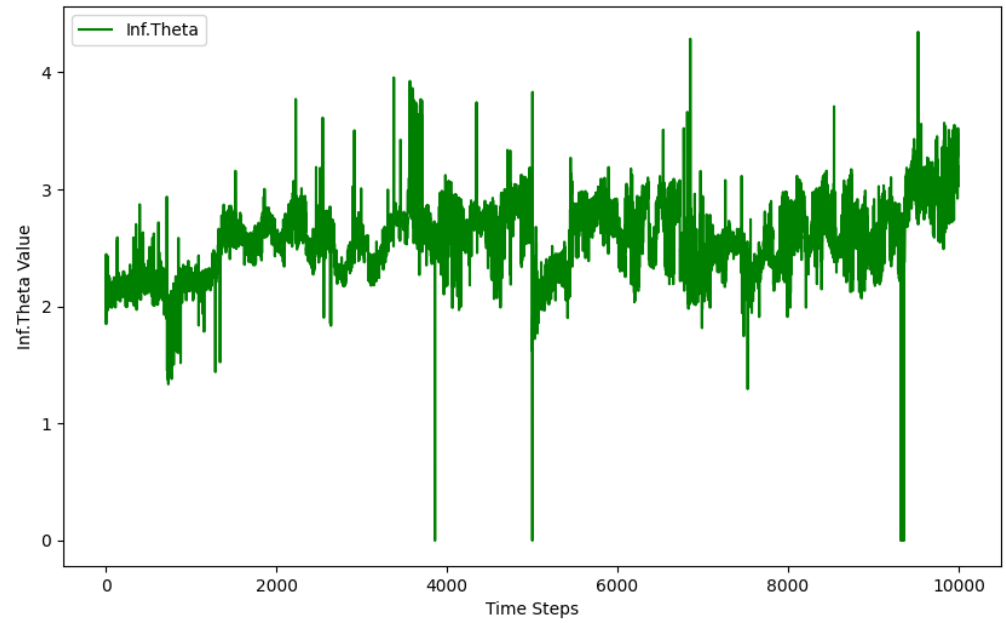

Figure 3.7: In USAir, the node BIL, Similarity

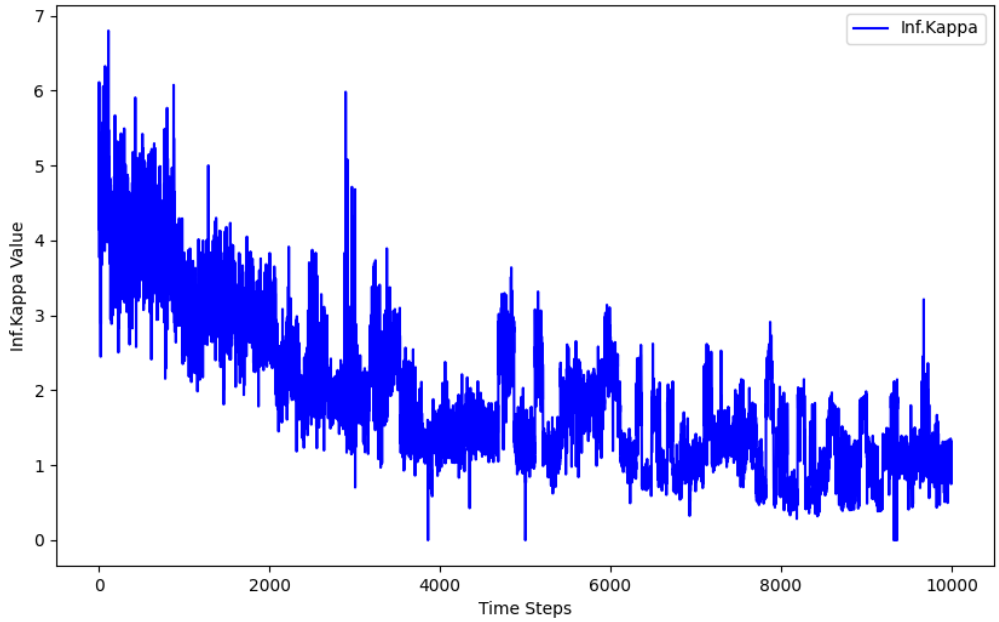

Figure 3.8: In USAir, the node BIL, Expected degree

In [1], the authors conduct a detailed analysis of the statistical characteristics of these trajectories. Specifically, they analyze the velocity increments, autocorrelation, variance, and the probability distribution of velocity increments, demonstrating the long-term dependence, stability, volatility, and the presence of abrupt changes or regularity in the time series. We extract the analysis of the similarity-popularity trajectories of CLT airports in the US airport network from the paper and provide a brief description here.As shown in Figure 3.9.

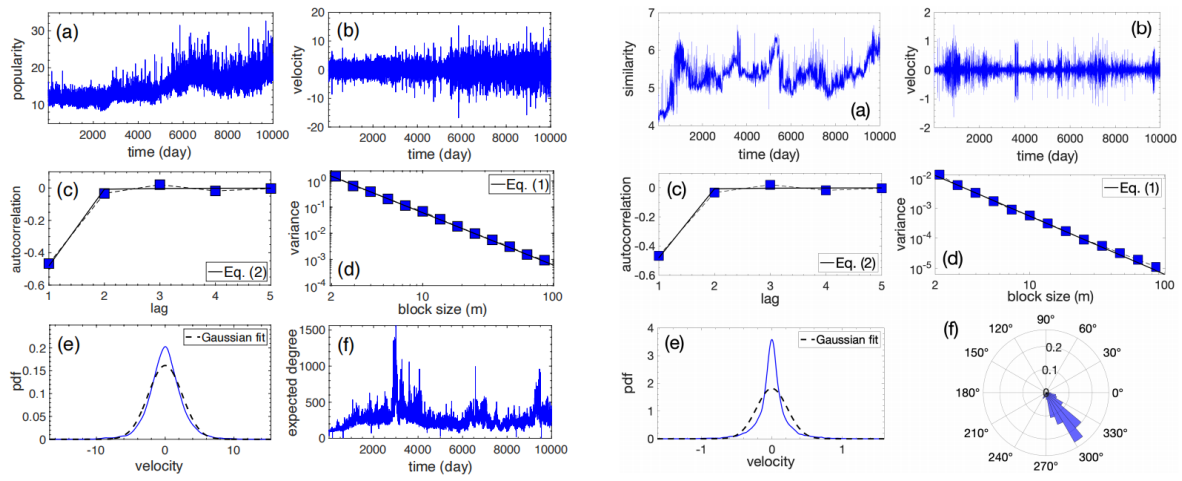

Figure 3.9: In USAir, the node CLT, popularity(a-left) and similarity(a-right) trajectories,Velocity Increment(b), Autocorrelation of the velocity process(c),Variance of the velocity process(d),Variance of the velocity process(e)[1].

(1) Velocity Increment of the Trajectory: From the graph, it can be seen that the velocity increments exhibit significant fluctuations throughout the entire time period, indicating that the time series experiences considerable instantaneous changes. This also suggests that the series has high non-stationarity

in the short term.

(2) Autocorrelation of the velocity process: The autocorrelation graph shows that the time series exhibits anti-persistence (negative autocorrelation) in the short term.Negative autocorrelation means that the velocity increments exhibit reverse correlation in the short term, meaning that the current change (e.g., an increase) tends to be followed by a change in the opposite direction (e.g., a decrease). This anti-persistence characteristic indicates that the original data often exhibits volatile patterns, such as a rapid increase at one point being followed by a decrease.

(3) Variance of the velocity process: The data in the graph show that the variance decreases as the block size increases, meaning that as the block size grows, the data's volatility gradually diminishes. This indicates that over larger time scales, the data's volatility tends to stabilize, suggesting that the series is more stable over longer periods.

(4) Velocity Increment Distribution: The probability distribution of the velocity increments in the graph shows clear symmetry and is close to a Gaussian distribution. This means that the velocity changes in the time series roughly follow a normal distribution. The black dashed line represents the Gaussian fit, showing that the distribution of velocity increments closely resembles the Gaussian distribution. This indicates that the velocity increments change relatively smoothly, with most of the variations concentrated within a smaller range, and the probability of extreme increments (very large positive or negative values) is low.

Through the analysis of these statistical characteristics, the time series appears non-stationary and highly volatile in the short term, exhibiting anti-persistence. However, over longer time scales, it shows some trend or persistence and exhibits a certain degree of stability.

Additionally, in the paper, the authors use the fractional Brownian motion model to obtain reasonable prediction results. It can be observed that the fractional Brownian motion model accurately predicts the long-term trend of the trajectory. Based on this, we attempt to use other methods to predict these trajectories. We focus not only on the prediction performance of the long-term trend of the trajectories but also on their predictive performance in terms of finer details.

3.3 Time series modeling

In time series prediction problems, the model can learn the temporal dependencies through historical data and capture the patterns of changes in the relative positions of the nodes. This approach avoids the complexity of directly handling graph structures, making the link prediction task more intuitive and easier to implement.

The commonly used methods for time series prediction include traditional methods, machine learning methods, and deep learning methods.

Traditional time series forecasting methods mainly rely on statistical models, with common ones including ARIMA, seasonal ARIMA, and exponential smoothing [63]. These methods assume that time series data exhibit certain statistical regularities, making them suitable for stationary data or data with seasonal and trend variations. ARIMA (AutoRegressive Integrated Moving Average) combines autoregressive (AR), differencing (I), and moving average (MA) components to capture dependencies in time series, es-

pecially for stationary data. Seasonal ARIMA (SARIMA) is an extension of the ARIMA model designed to handle time series with seasonal fluctuations, accounting for periodic changes in the data. Exponential smoothing methods predict by assigning different weights to historical data, allowing them to capture trends and seasonal variations [63]. The advantage of traditional methods lies in their simplicity and ease of implementation, but they perform poorly when dealing with complex, nonlinear, or long-term dependency data.

With the rapid development of machine learning, machine learning methods have been widely applied in time series forecasting. Common machine learning methods include Random Forest [64], Support Vector Machine (SVM) [65], and XGBoost [66], among others. These methods transform time series data into supervised learning problems through feature engineering, using machine learning models for regression or classification tasks. Random Forest [64]predicts future values of time series by training multiple decision trees and performing voting, and it can handle nonlinear relationships without strong assumptions. SVM [65]predicts by constructing hyperplanes in high-dimensional space, making it suitable for small datasets or complex features. XGBoost [66], an ensemble method based on gradient-boosted trees, can handle complex nonlinear relationships and has performed excellently in many time series forecasting tasks. Machine learning methods perform well in capturing nonlinear and complex relationships, but they typically require extensive feature engineering and struggle to directly address long-term dependencies in time series data.

Deep learning methods have made significant progress in time series forecasting, especially in handling long sequences of data and complex nonlinear relationships. Common deep learning methods include LSTM (Long Short-Term Memory networks) [67], GRU (Gated Recurrent Units) [68], and Transformer [69]. LSTM [67]is an improved version of Recurrent Neural Networks (RNN) [70], which addresses the issue of long-term dependencies through gating mechanisms, making it suitable for forecasting long time series data. GRU [68], similar to LSTM [67], has a simplified structure and faster training speed, yet it can achieve similar results as LSTM [67]in many tasks. Transformer [69], based on the self-attention mechanism, captures dependencies between different time steps in a sequence, making it particularly suitable for forecasting multivariate time series. It also allows for parallel processing of the entire sequence, making it computationally efficient. Deep learning methods can automatically extract features from data, handle complex nonlinear relationships and long-term dependencies, and are suitable for large-scale datasets, but they typically require large amounts of data and computational resources.

3.4 Method of prediction

In this task, we focus more on the trend changes of popularity-similarity trajectories over long time periods. In recent years, with the development of deep learning, neural network methods have been widely applied to time series analysis and forecasting tasks. The advantages of deep learning methods in long-term forecasting mainly lie in their powerful nonlinear modeling ability, automatic feature extraction capability, and ability to handle large-scale data. Compared to traditional statistical methods, deep learning models can effectively capture the complex nonlinear relationships in the data, which is particularly important for long-term forecasting tasks, as the long-term trends and patterns in the data often exhibit highly nonlinear characteristics. Furthermore, deep learning methods can automatically learn meaningful

features from the data, without relying on manually designed features, which allows the model to adapt flexibly to different types of data. For time series data, deep learning models are capable of handling dependencies in long sequences, and they excel in capturing long-term trends and periodic fluctuations. Architectures like Long Short-Term Memory networks (LSTM) [67] and Gated Recurrent Units (GRU) [68]can maintain effective memory of historical information over long time spans and predict future trends.

In addition to their modeling capabilities, deep learning models also exhibit strong robustness when handling noisy and uncertain data. They can automatically extract underlying patterns from large amounts of data, even in the presence of irregularities and disturbances, thus improving the accuracy of predictions. Especially when facing complex systems (such as climate change, energy demand forecasting, etc.), deep learning models can effectively integrate information from different sources, perform multimodal learning, and combine external features (such as economic indicators, weather changes, etc.), further enhancing the reliability of long-term predictions. Deep learning also has strong scalability, and with the improvement of computational power, these models can handle increasingly large datasets and continue to optimize through ongoing training, gradually improving the accuracy of long-term forecasts.

Therefore, deep learning methods show a clear advantage over traditional methods in long-term forecasting, particularly in capturing complex patterns and handling large-scale datasets. In addition, the adaptive learning ability of deep learning models allows them to automatically adjust as the data changes, further improving prediction performance. Despite challenges such as error accumulation and model stability in long-term forecasting, deep learning methods can still provide accurate predictions in many application scenarios, especially when dealing with data influenced by nonlinearity, periodicity, and multiple factors, demonstrating their unique advantages.

This study adopts the Transformer architecture as the core model for time series forecasting. Originally proposed by Vaswani et al. in 2017 in the paper ”Attention is All You Need,” [69]the Transformer was first introduced for natural language processing (NLP) tasks, particularly machine translation and text generation. In contrast to traditional recurrent neural networks, the key innovation of the Transformer lies in its full reliance on the self-attention mechanism, which enables the model to capture dependencies between any two positions in a sequence, regardless of their distance.This is a standard Transformer architecture, which adopts an encoder–decoder structure.

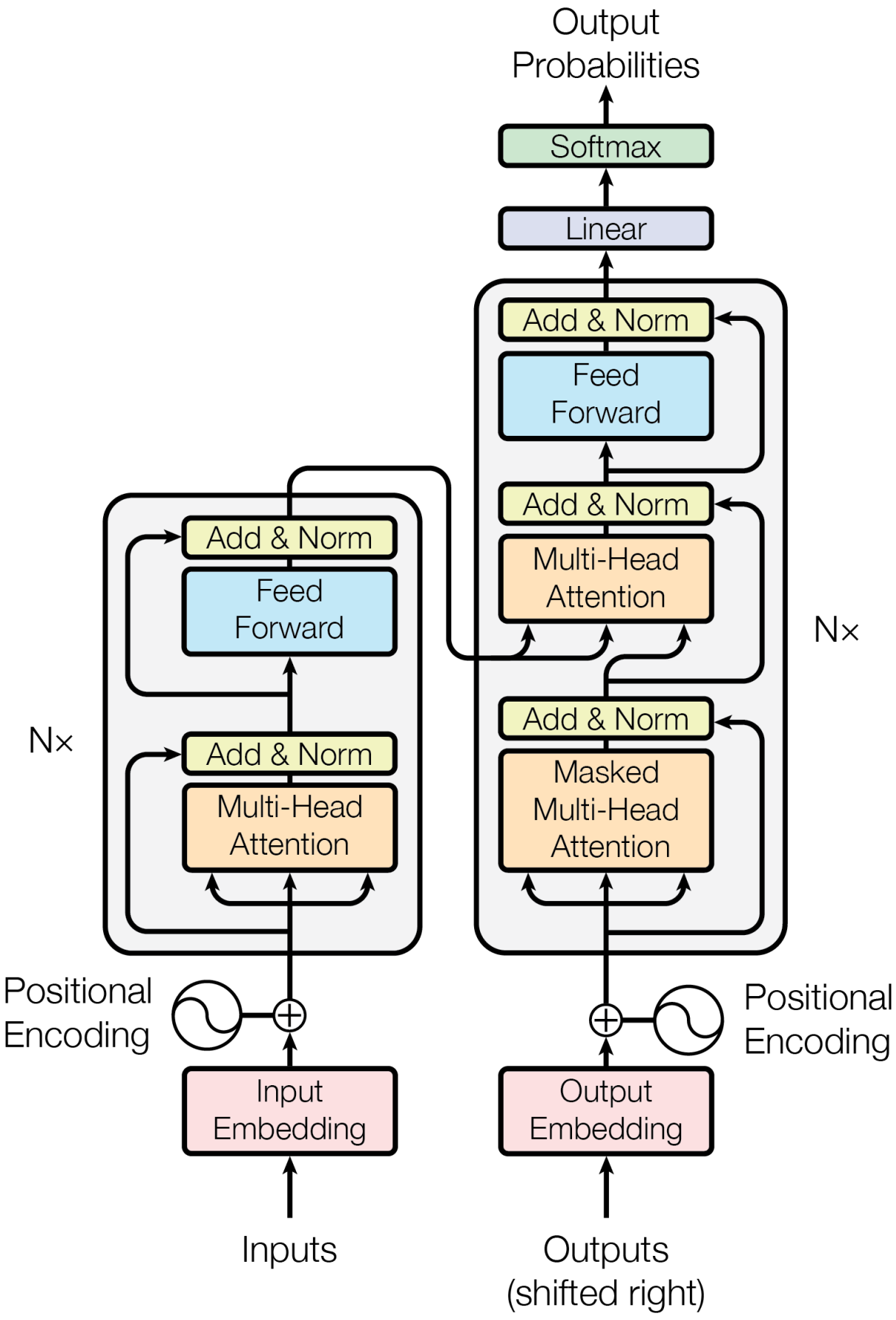

Figure 3.10: Standard Transformer structure diagram

By computing attention weights in parallel, the Transformer avoids the sequential nature of RNNs, thereby significantly improving computational efficiency, parameter control, and training speed. The architecture incorporates multi-head attention to capture diverse relationships in different subspaces, and employs positional encoding to introduce sequential information into the model, compensating for the lack of inherent position awareness. Additionally, the Transformer includes components such as feed-forward neural networks, layer normalization, and residual connections, forming a modular and extensible deep network framework.

In recent years, due to its powerful sequence modeling capability, the Transformer has been increasingly introduced into the field of time series forecasting. Its core mechanism—the self-attention mechanism—enables parallel modeling of all time steps within a sequence, effectively capturing both short-term and long-term temporal dependencies. This feature overcomes the limitations faced by traditional recurrent neural networks, such as RNNs [70]and LSTMs [67], which often suffer from vanishing gradients and low training efficiency when modeling long-term dependencies.

One of the earliest studies to apply Transformer to time series forecasting is the work of Li et al. (2019), who proposed the LogSparse Transformer to enhance temporal locality and reduce the memory overhead inherent in the original architecture [71]. Their work demonstrated that Transformer could outperform traditional recurrent models when properly adapted for temporal data.Following this, Lim et al. (2021) introduced the Temporal Fusion Transformer (TFT), which combines self-attention with recurrent units and gating mechanisms for interpretable multi-horizon forecasting [72]. The TFT was among the first models to offer a hybrid approach tailored specifically for complex real-world forecasting scenarios, such as electricity demand and retail sales.Lara-Benítez et al. (2021) conducted a systematic evaluation of the standard Transformer architecture for univariate time series forecasting, comparing its performance with LSTM and CNN models [73]. Their findings confirmed that, although Transformers are not always superior in all settings, they show promising results in capturing long-term dependencies and nonlinear patterns in temporal data.In the financial domain, Lezmi and Xu (2023) applied Transformer models to asset management and investment forecasting, highlighting the architecture's effectiveness in modeling highly volatile and multivariate financial time series [74].

These early works have collectively laid the foundation for a growing body of research exploring Transformer-based models in time series tasks. They also underscore the importance of architectural adaptation—such as attention sparsity, position encoding, and hybrid modeling—for the successful deployment of Transformer models in time series forecasting.

Figure 3.11 illustrates the steps of time series modeling and prediction in this experiment. The left side of the figure shows the process of preparing the time series data, which has been described in detail earlier. The figure also outlines the specific experimental procedure and explains how the Transformer model processes the data.

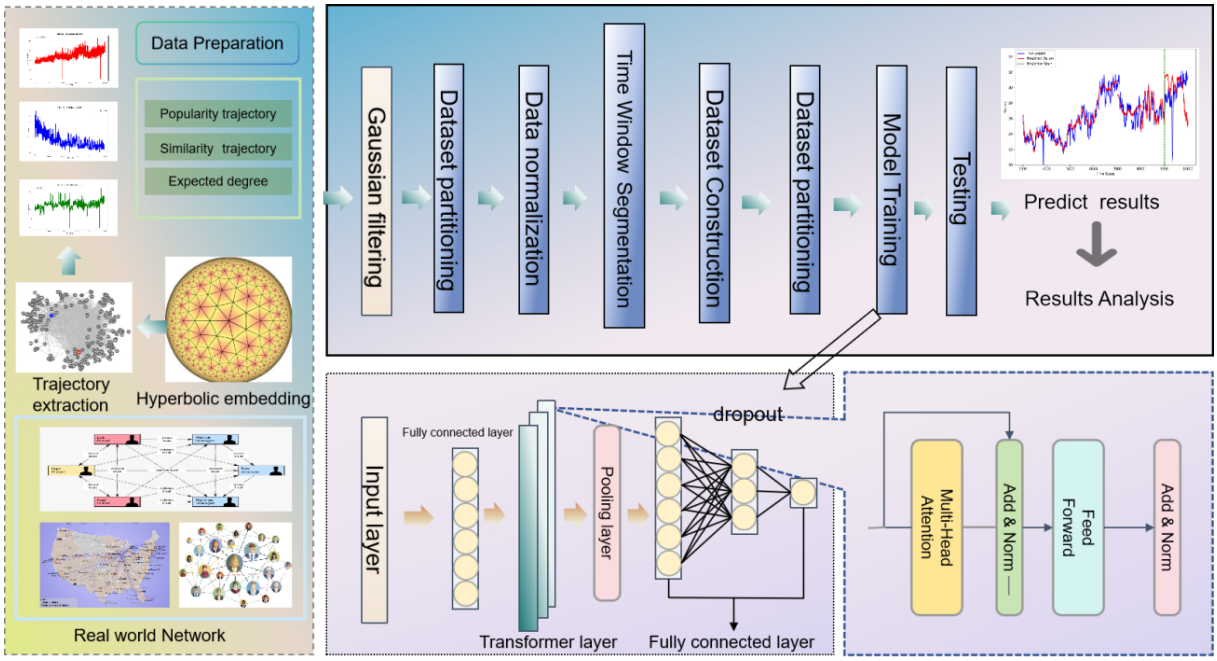

Figure 3.11: Experimental process.

This study is built upon the standard Transformer [69]architecture, with several targeted structural adaptations to better address the characteristics of time series forecasting tasks. Specifically, we adopt only the encoder component of the original Transformer [69]and omit the decoder module. The decoder was originally designed for autoregressive text generation tasks in natural language processing . However, in time series forecasting—especially under the direct forecasting setting—the objective is to map historical observations to future time steps, rather than generating outputs sequentially. Therefore, it is more appropriate to employ an encoder-only design, which directly extracts temporal representations from historical inputs and performs forecasting through subsequent feedforward layers.

This modeling strategy—using only the encoder for time series modeling—has been widely adopted in recent literature. For instance, Yong Liu et al. proposed the General Time Transformer (GTT) [75], a foundational encoder-only model capable of zero-shot multivariate time series forecasting. Through pretraining, GTT learns general temporal representations and achieves prediction accuracy close to or even surpassing that of supervised models on multiple public datasets.

Building on this idea, the W-Transformer model, proposed by Lena Sasal et al. [76], is specifically designed for non-stationary univariate time series. By combining the Maximal Overlap Discrete Wavelet Transform (MODWT) with a Transformer encoder structure, the model performs decomposition of the input series and applies local attention modules on the resulting sub-series. This allows for effective modeling of non-stationarity and long-range nonlinear dependencies, demonstrating the structural flexibility of the encoder in complex time series settings.

Additionally, Yong Liu et al. introduced the iTransformer model [77], which adopts an inverted encoder structure that applies attention and feedforward operations along the transformed feature dimensions. This design focuses on capturing global dependencies in multivariate time series and has achieved state-of-the-art performance on multiple real-world datasets, further validating the effectiveness and adaptability of encoder-only architectures in temporal modeling.

In summary, these representative studies strongly support the feasibility and effectiveness of using encoder-only structures for time series forecasting, offering a lightweight, efficient, and highly extensible approach to Transformer-based temporal modeling.

3.4.1 Experimental setup

3.4.1.1 Data preprocessing

The original data contains a large amount of high-frequency noise, and the model needs to handle these fluctuations during training, which increases the difficulty of learning. Noise and short-term fluctuations can distract the model's attention, making it harder for the model to learn the long-term trends. Firstly, noise can obscure effective trends. In time series data, high-frequency noise causes drastic fluctuations in the data, interfering with the long-term trends and patterns. When the model encounters such noise during training, it may mistakenly treat the noise as an important signal, leading to difficulties in learning and possibly preventing the accurate capture of the true trend of the data. Therefore, removing the noise helps the model focus on the main trend of the data, improving the prediction accuracy.

After filtering, the high-frequency noise in the data is removed, leaving mainly the long-term trend information. The learning goal for the model becomes clearer. This allows the model to focus more quickly on learning the main features of the data without getting "lost" in the noise. Learning becomes easier and more efficient because the data becomes smoother and focuses on the long-term trend, enabling the model to identify important patterns in the data more quickly.

Gaussian filtering is a commonly used denoising technique in time series analysis. Its main purpose is to smooth the data and remove high-frequency noise in the time series, while preserving the long-term trends and low-frequency components. This filtering method is based on a weighted averaging approach using the Gaussian distribution, where each data point is averaged with its surrounding points to reduce the impact of short-term fluctuations and highlight the long-term trend of the data. In Gaussian filtering, each data point is weighted by a Gaussian distribution function, with points closer to the current data point receiving higher weights and those farther away receiving lower weights, thus achieving a smoothing effect.

The mathematical principle of Gaussian filtering can be represented by the Gaussian function, with the following formula:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \tag{3.5}$$

where $G(x)$ is the Gaussian filter function, representing the weight at the position x. σ is the standard deviation of the Gaussian distribution, which determines the degree of smoothing. Smaller values of σ produce smaller smoothing effects, preserving more of the original data features, while larger values of σ result in a more pronounced smoothing effect, reducing more fluctuations and noise. When applying Gaussian smoothing, the first step is to select a window size w, which represents the neighborhood range around each data point. Then, the weighted average is computed for each data point, considering the weights of its neighboring points, to obtain the smoothed data.

By adjusting the standard deviation of the Gaussian distribution, the degree of smoothing can be flexibly

controlled. Smaller values of σ preserve more details, while larger values of σ provide stronger smoothing effects, adapting to different application scenarios. In addition, it is worth noting that Gaussian filtering has a minimal impact on boundary effects. Compared to other filtering methods (such as mean filtering), it uses a smooth weighted window to process the data, thereby avoiding the excessive smoothing caused by boundary effects.

The following figures show the filtered data with different degrees of smoothness controlled by different values of σ. Specifically, the larger the σ, the smoother the result. We present the filtering effects for σ values of 1, 5, and 10, respectively. Figure 3.12 to Figure 3.15 display the filtering effects on the BIL node in the USAir dataset. Figures Figure 3.16 to Figure 3.19 show the filtering effects on node 10 in the arXiv dataset. Figure 3.20 to Figure 3.23 present the filtering effects on node 0x0A2F87E5 in the PGP dataset.

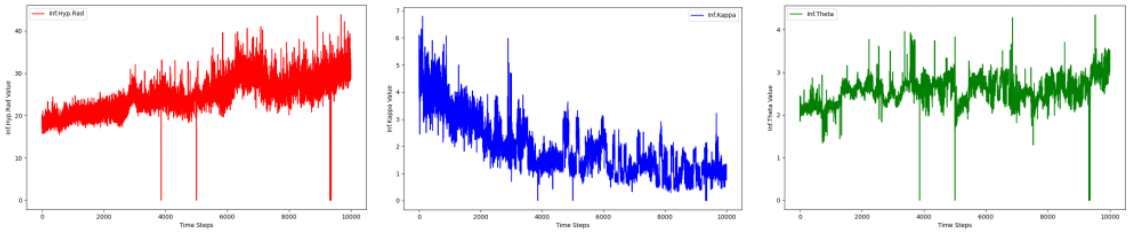

Figure 3.12: USAir, the node BIL, Original trajectories.

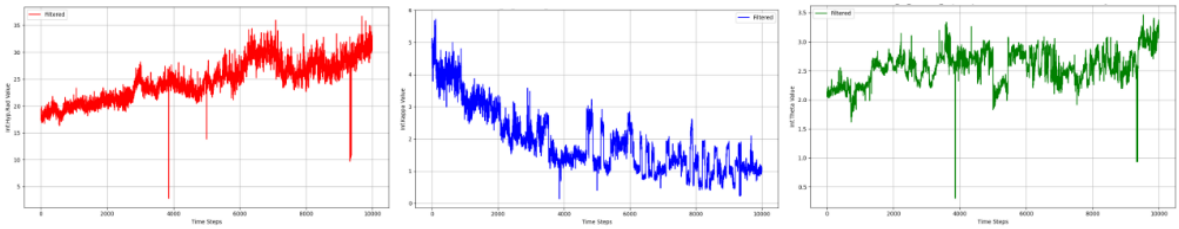

Figure 3.13: USAir, the node BIL, The filtered trajectories when $\sigma = 1$.

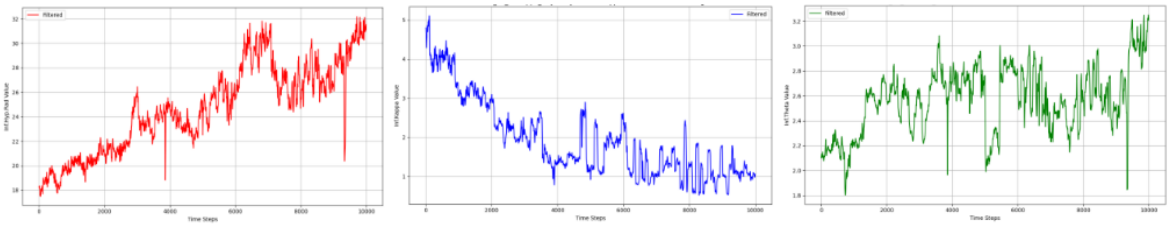

Figure 3.14: USAir, the node BIL, The filtered trajectories when $\sigma = 5$.

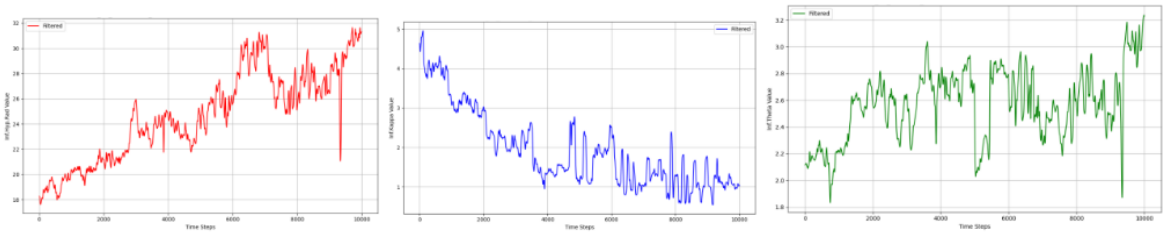

Figure 3.15: USAir, the node BIL, The filtered trajectories when $\sigma = 10$.

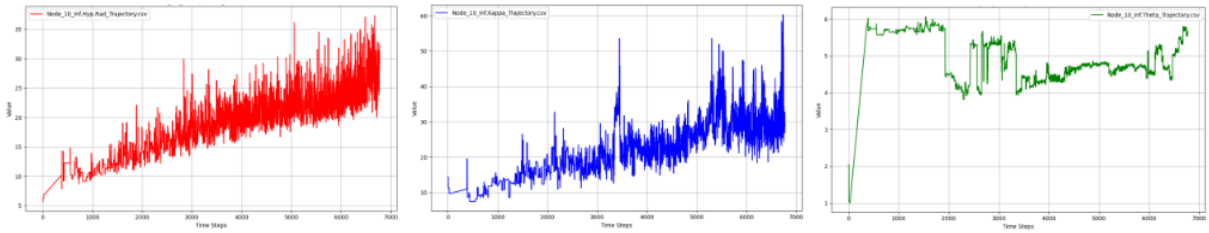

Figure 3.16: arXiv, the node 10, Original trajectories.

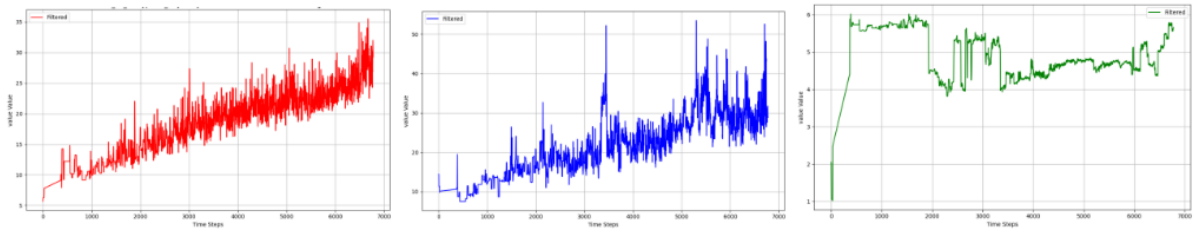

Figure 3.17: arXiv, the node 10, The filtered trajectories when $\sigma = 1$.

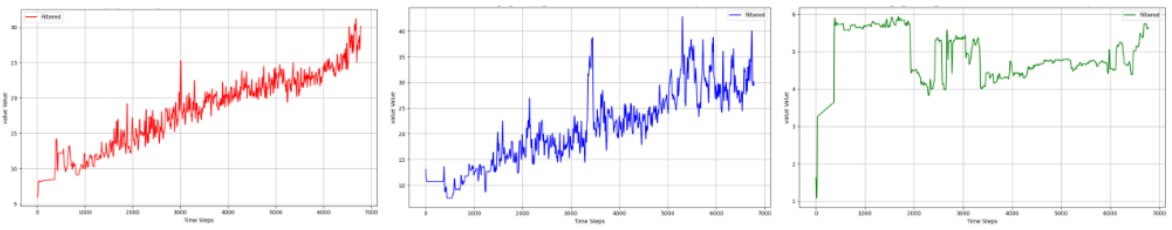

Figure 3.18: arXiv, the node 10, The filtered trajectories when $\sigma = 5$.

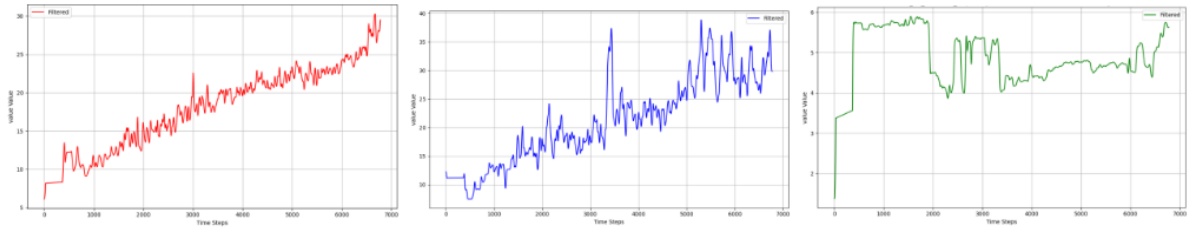

Figure 3.19: arXiv, the node 10, The filtered trajectories when $\sigma = 10$.

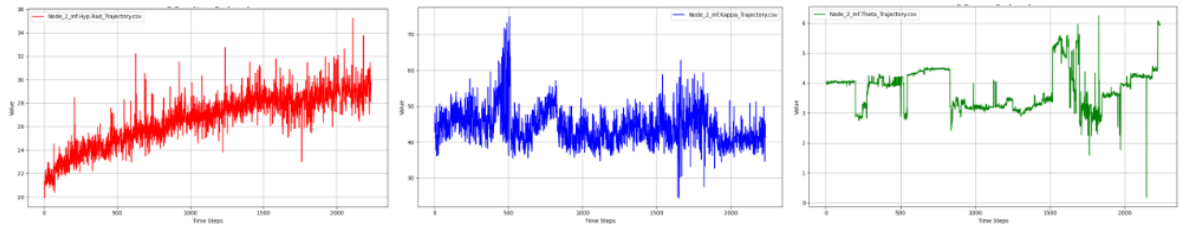

Figure 3.20: PGP, the node 0x0A2F87E5, Original trajectories.

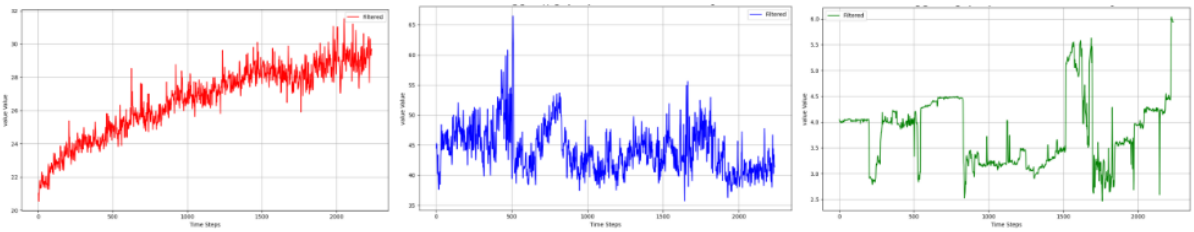

Figure 3.21: PGP, the node 0x0A2F87E5, The filtered trajectories when $\sigma = 1$.

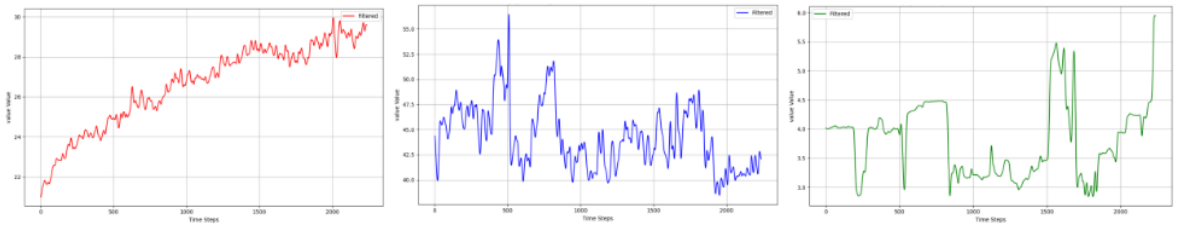

Figure 3.22: PGP, the node 0x0A2F87E5, The filtered trajectories when $\sigma = 5$.

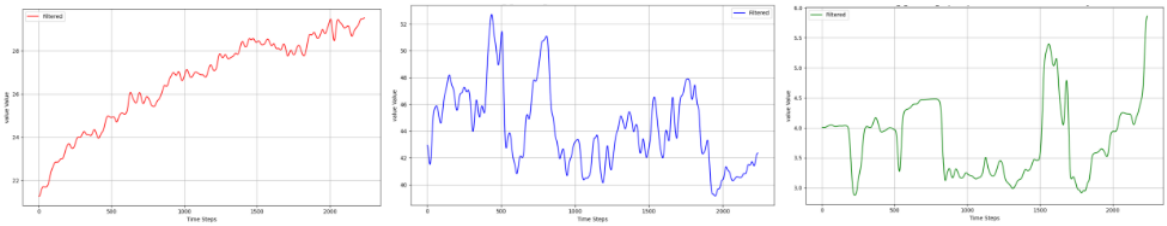

Figure 3.23: PGP, the node 0x0A2F87E5, The filtered trajectories when $\sigma = 10$.

In this research, we selected the data processed with σ = 5 as the input data.

3.4.1.2 Dataset partitioning

For the American airport data, use the first 10,000 data points, with the first 9,000 points from the trajectory as the training set, and the last 1,000 points as the test set. For the arXiv data, use data from the 500th to the 6500th point, with the first 6,000 points for training and the next 500 points for testing. For the PGP data, use the first 2,000 data points, with the first 1,600 points as the training set and the last 400 points as the test set.

3.4.1.3 Data normalization

If the data is not normalized, features with larger value ranges may dominate the model's learning process, preventing the model from effectively capturing information from other features. When using gradient descent optimization, unnormalized data can cause uneven gradient updates, where certain features have an exaggerated influence, leading to weight updates that overly rely on some features while neglecting others. Additionally, unnormalized data may slow down the model's convergence speed, or in some cases, cause it to fail to converge altogether, because the scale differences between features can make the optimization process unstable. Therefore, normalization helps improve the efficiency and stability of the model's training process.

Many models based on gradient descent optimization (such as neural networks, support vector machines, etc.) rely on gradient calculations during training. When the range of input data differs significantly, the gradient changes will also vary, causing some features' gradients to change too much or too little, which can affect the model's convergence speed. By normalizing the data, all values are scaled to a similar range, helping the gradient descent algorithm to find the optimal solution more quickly, thus accelerating the model's convergence.

Normalization also helps avoid numerical overflow or underflow. If the input data has a very large range, it can lead to numerical overflow (for example, gradients being too large) or underflow (for example, very small numbers causing results to be zero) during the computation process. In such cases, the model's training becomes unstable and may not yield effective predictions. Normalization compresses the data into an appropriate range, helping to avoid these numerical issues and ensuring the stability of the training process.

In our experiment, the MinMaxScaler normalization method is used. MinMaxScaler is a commonly used normalization technique that linearly scales the data to a specified minimum and maximum range. This method is particularly suitable when the data distribution is known and does not contain significant outliers. After normalization, the range of each dataset is between 0 and 1.

For each data point x, the normalized value x' can be calculated using the following formula:

$$x' = \frac{x - \min}{\max - \min} \tag{3.6}$$

3.4.1.4 Time window partitioning

In time series prediction, one important reason for windowing is that models cannot directly handle very long sequences and are highly dependent on a large number of samples.

First, deep learning models face computational and memory bottlenecks when dealing with long time series. Traditional time series data often contain a large number of time steps, and if the entire sequence is used as a single input for training, the model needs to handle a huge amount of input data. This not only results in a large computational cost but also makes the training process very slow. Furthermore, deep learning models, especially recurrent neural networks (RNNs) or long short-term memory networks (LSTMs), encounter issues such as vanishing or exploding gradients when processing long sequences. The model struggles to capture the dependencies between time steps that are far apart in the sequence.

Therefore, directly using long sequences as inputs is highly inefficient.

Second, deep learning models rely on a large number of samples to effectively learn the patterns in the data. In time series data, multiple samples are typically needed for the model to learn stable patterns and trends. However, if long time series are used without windowing, the model may not obtain enough samples for training, which could lead to an unstable learning process or underfitting. For example, if the first 8,000 data points are directly used as historical input to predict the subsequent 2,000 data points as the target, then for each node, the entire trajectory can only be used to construct a single training sample. However, deep learning methods fundamentally rely on extracting underlying patterns from a large number of samples, learning effective representations and prediction functions in a data-driven manner. When the number of samples is severely limited, the model struggles to learn features with sufficient generalization capability, which significantly compromises its performance on complex time series forecasting tasks. Therefore, naively dividing the full trajectory into a single input–output pair fails to meet the basic data volume requirements necessary for training deep learning models.

By dividing long sequences into multiple smaller time windows, the model can extract independent training samples from these windows, allowing it to use more training data and thus improve the learning effectiveness.

Additionally, much of the information in long sequences may not be relevant for prediction, and windowing helps the model focus on useful local information. For example, the trend of changes in a certain period of history may be helpful for current predictions, while the long-term trend or noise in the full sequence may have little impact on the prediction. Windowing, by limiting the amount of data input at a time, helps the model focus on the most relevant parts, thus improving its accuracy.

In summary, windowing effectively addresses the problem of models not being able to directly handle long sequences and provides enough samples for the model to learn. By appropriately setting the window size, it can control the computational load while enhancing the model's training effectiveness and generalization ability.

In this experiment, the window length is set to be the same as the target sequence length. For example, in the airport data, the window length is set to 1000, and the target sequence length is also set to 1000. Each sample pair consists of 1000 time steps of historical data and 1000 time steps of prediction data. After one sample is formed, the sliding window moves one step forward to create a new sample, continuing until the entire training set is processed. For the arXiv dataset, the window length is set to 500, and for the PGP dataset, the window length is set to 400.

3.4.1.5 Dataset construction

Based on the time window division, the input data X and prediction label data Y are partitioned. X represents the historical data length determined by the time window, while Y is the length of the data to be predicted. Each pair of (X, Y) forms a training sample.Figure 3.24 illustrates the process of constructing sample pairs in our study.

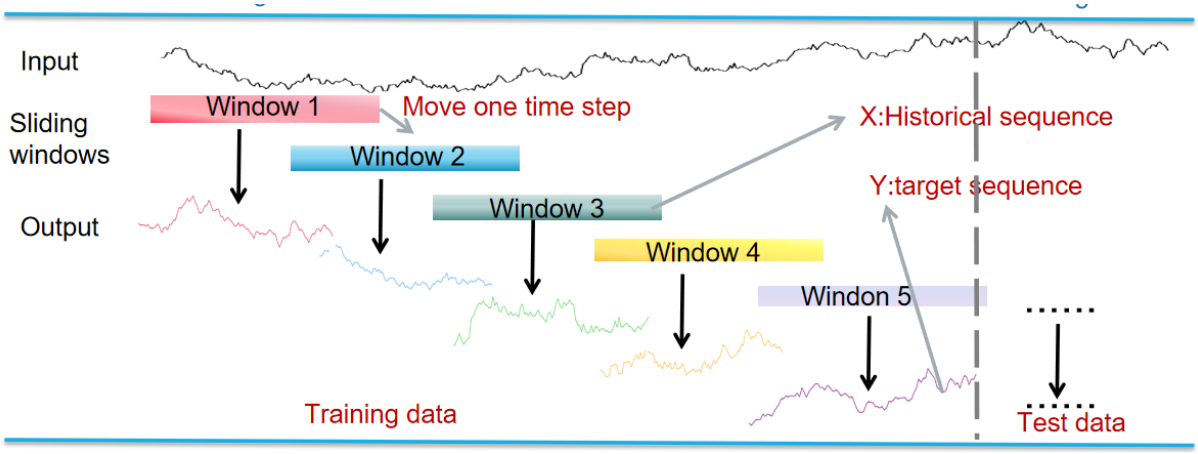

Figure 3.24: Dataset construction process.

3.4.1.6 Model training

In time series forecasting, the primary objective of model training is to learn the relationship between input data X and target data Y, enabling the model to make accurate predictions for future time steps. Specifically, the model learns the following:

(1) Pattern recognition: The model identifies regularities, trends, seasonal patterns, and periodic changes in the time series data. By observing historical data (X), the model attempts to capture the mapping between input sequences and target prediction values (Y).

(2) Temporal dependencies: Time series data exhibit temporal dependency, meaning that data from current time steps influence future values. The model learns these temporal dependencies to accurately forecast upcoming values.

(3) Feature transformation and representation: Through the hierarchical structure of the model (e.g., multilayer perceptrons or convolutional layers in neural networks), the input data X are transformed into more abstract and meaningful feature representations. These features help the model better understand the underlying relationships within the data.

3.4.1.7 Testing

Use the trained model to make predictions and collect the prediction results for both the training and testing datasets.

3.4.1.8 Result evaluation

At the same time, evaluate the prediction results for both the training and testing sets using multiple metrics to assess the model's performance on different data and trajectories.

3.4.2 Experimental principles

In our study, we made adaptations to the standard Transformer to better suit our task. Specifically, we only used the encoder part and the attention mechanism from the original architecture — we did not

include the decoder, which is mainly used for text generation. In addition, we added fully connected layers, pooling layers, and dropout layers before and after the encoder.

These components help handle non-text inputs, support dimensional transformations within the network, enhance the model's expressiveness, and reduce the risk of overfitting.The specific architecture of the Transformer model used in this research is as follows:

3.4.2.1 Input Layer

The input layer is the first layer of a neural network. Its main function is to receive external input data and pass it to the subsequent network layers. The input data is received through the Input layer.The input layer receives a univariate time series with a shape of (maxlen, 1), where maxlen denotes the length of the sequence and is determined by the size of the historical data (i.e., X) used in each sample. In the US Air dataset, maxlen is set to 1000; in the arXiv dataset, it is 500; and in the PGP dataset, it is 400. The dimensionality of 1 reflects the absence of external features in these time series, meaning that only a single variable is observed over time.

The standard Transformer is typically designed to process discrete word data, where the input is usually represented as a one-dimensional sequence of length maxlen, denoting a token sequence of that length. In contrast, time series data generally consist of continuous real-valued observations that reflect the variation of a variable over time. Even for univariate time series, the original input shape is commonly (maxlen, 1), where the dimension 1 represents the feature dimension. This additional dimension captures the feature space structure of the time series, indicating that each time step may carry one or more real-valued features. In multivariate settings, the input shape is further extended to (maxlen, num_features), where each time step contains multiple feature values.

Therefore, compared to token sequences, time series inputs have an explicit feature dimension in their input shape. This structural difference necessitates the use of dense layers to embed the real-valued inputs into higher-dimensional representations, rather than relying on an embedding lookup table, as is common in NLP applications.

3.4.2.2 Fully Connected Layer

The input data passes through a fully connected layer. In this study, to meet the input requirements of the Transformer architecture, we introduce a fully connected layer (Dense(64)) after the original time series input, mapping each time step's real-valued feature into a 64-dimensional vector to construct an embedded sequence. This operation is referred to as "time series embedding" and serves a similar role to the embedding lookup applied to discrete token IDs in standard Transformer models.

This step is essential because the core module of the Transformer—multi-head self-attention—requires the input to be a sequence of vectors (typically with shape (batch_size, seq_len, d_model)) in order to compute dependencies and similarities across time steps. Directly feeding the raw time series (e.g., with shape (batch_size, seq_len, 1)) into the model may be syntactically valid, but the extremely low information dimensionality limits the model's ability to extract meaningful features and capture complex patterns.

In contrast, projecting each scalar input into a higher-dimensional embedding space via a Dense layer significantly enhances the model's expressive capacity. It enables the attention mechanism to operate

on richer feature representations and better capture temporal dynamics and nonlinear structures, thereby providing a stronger foundation for subsequent representation learning and prediction.

3.4.2.3 Transformer Layer

This component consists solely of the encoder part from the standard Transformer architecture. For this encoder, we did not modify its structure but only invoked it and adjusted its parameters. The Transformer layer serves as the core of the entire model, responsible for extracting features from the time series.

This part is constructed by stacking multiple TransformerBlocks to form the main functional module of the model. Each TransformerBlock contains a multi-head self-attention mechanism and a feed-forward neural network.In this model, the Transformer layer serves as the core component of the entire network. Its primary function is to model the dependencies and dynamic variations across different time steps in the time series. Built upon the self-attention mechanism, the Transformer layer is capable of directly capturing interactions between any two positions in the input sequence. This enables the model to effectively learn long-term dependencies, while avoiding the vanishing gradient problem commonly encountered in traditional recurrent architectures when modeling long sequences. Such a mechanism is particularly crucial for time series data, where dependencies often span multiple time steps and exhibit nonlinear or periodic behaviors.

A Transformer layer typically consists of two sub-modules: Multi-Head Self-Attention and a Feed-Forward Network (FFN). The multi-head attention mechanism allows the model to extract multi-scale features from different subspaces, while the feed-forward network further enhances the representation at each time step. These two components are connected via residual connections and layer normalization, which help improve model stability and training efficiency.

In terms of data flow, the input to the Transformer layer in this study is generally a tensor of shape (batch_size, maxlen, 64), where 64 represents the embedding dimension at each time step. After being processed through the multi-head attention and feed-forward sublayers, the output tensor retains the same shape (batch_size, maxlen, 64). This architectural design allows the model to process all time steps in parallel, improving computational efficiency while preserving contextual representations for each time step.

(1) Attention Mechanism: The primary function of the attention mechanism is to compute similarity-based weights between each time step and all other time steps within the historical sequence (X), and to use these weights to aggregate information accordingly. The output of this weighted aggregation is not the final prediction, but rather a new contextual representation vector that encodes the relevance between the current time step and the entire historical sequence. This vector is typically passed into subsequent feed-forward neural network layers. Structurally, the attention mechanism performs a reweighting and enhancement of each time step's representation without altering the original sequence length or feature dimension.

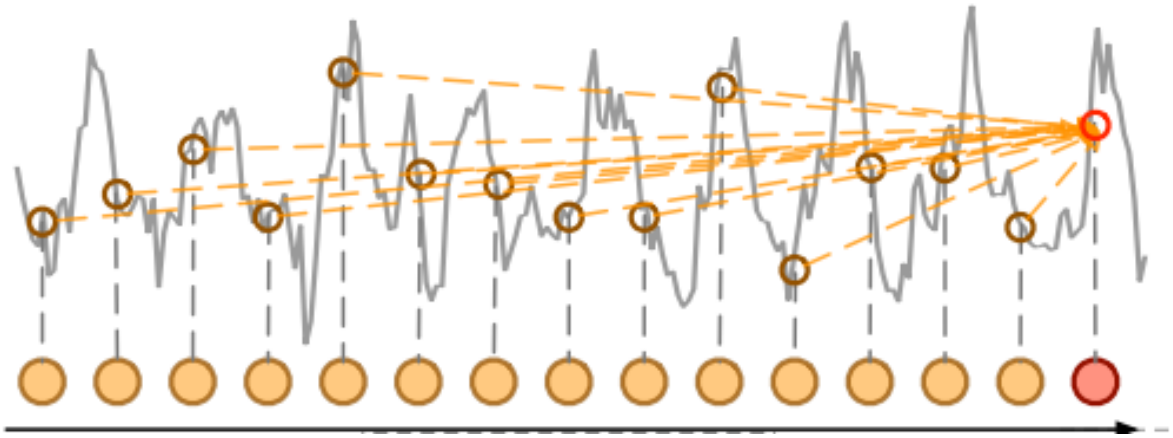

Figure 3.25: Illustration of the Attention Mechanism [78]

The figure 3.25 [78] illustrates the computation process of the attention mechanism. The current time step (represented by the red node on the right) serves as the query position (Query, Q), and its representation is compared against every time step in the historical sequence (orange nodes). Each of these time steps generates a key vector (Key, K) and a value vector (Value, V). The model computes the similarity between the query and each key—typically using a dot product followed by scaling and a softmax function—to produce a set of normalized attention weights. These weights indicate the degree to which the current time step attends to information from each historical time point. Finally, the model aggregates all the value vectors from the historical time steps using these attention weights, yielding an output representation for the current time step. This representation integrates dynamic information from multiple time steps, enabling the model to capture long-range dependencies, recurring patterns, or abrupt changes in the time series and to form a context-aware encoding tailored to the forecasting task.

The following is a detailed computational procedure for the attention mechanism. In the self-attention mechanism, the calculation is as follows:

$$\text{Attn}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V \tag{3.7}$$

where QK^T represents the dot product of the query and key, which measures the compatibility between the query and all keys. $\frac{1}{\sqrt{d}}$ is a scaling factor used to prevent the dot products from becoming too large. The `softmax` function is used to normalize these values, ensuring that the attention scores sum to 1 and obtain the weights for each value.Then, these weights are applied to the corresponding values (V), and the final output is the weighted sum.

This study extends the basic attention mechanism to a multi-head attention mechanism. In multi-head attention, the representation of each time step is processed in parallel by multiple attention heads, generating corresponding contextual representations. Specifically, each time step in the input sequence (represented as a 64-dimensional vector) is projected through multiple linear transformations to produce different sets of queries, keys, and values for each head. Each head independently performs a scaled dot-product attention calculation and outputs its own attention result. These out-

puts are then concatenated along the feature dimension and passed through a final linear layer to map the result back to the original dimension, forming the overall output of the multi-head attention module. The entire computation is performed independently and in parallel across all heads, and the outputs are unified at the final stage. Compared to single-head attention, the multi-head structure allows the model to capture richer and more diverse information from the sequence. Figure 3.26 [79]illustrates the complete structure of multi-head attention and the internal computation process of scaled dot-product attention.

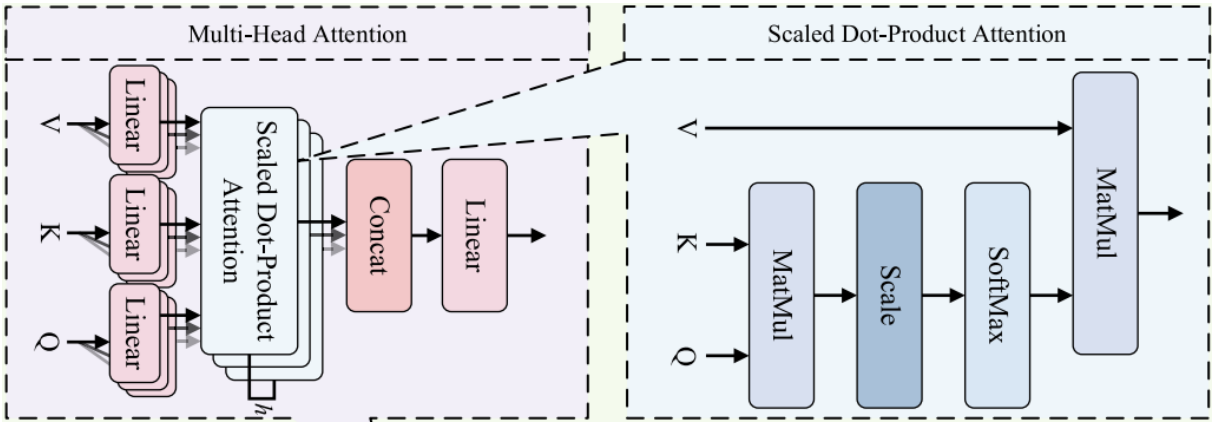

Figure 3.26: Architecture of Multi-Head Attention and Scaled Dot-Product Attention[79]

Assuming there are h heads, the input queries Q, keys K, and values V are mapped into the query, key, and value subspaces for each head, respectively:

$$\text{head}_p = \text{Attn}(QW_p^Q, KW_p^K, VW_p^V) \tag{3.8}$$

where QW_p^Q, KW_p^K, VW_p^V are the parameter matrices for each head, used to generate different subspace representations.In most existing studies, the number of attention heads commonly used in multi-head attention is 4 or 8. In this study, we adopt a 4-head attention mechanism. It is worth noting that the dimensionality of the time series embedding (set to 64 in this study) must be divisible by the number of attention heads.

Each head computes a unique attention value, and finally, the results of all heads are concatenated.

(2) Residual Connection:In In the standard Transformer encoder, residual connections appear twice: once after the multi-head attention module and once after the feed-forward network. Their purpose is to stabilize training, alleviate gradient vanishing, and enhance feature representation capability. In this study, we retain this structure without any modification.

Specifically, assuming the input is x, the output after passing through a sub-layer is given by:

$$\text{Output} = \text{LayerNorm}(x + \text{SubLayer}(x)) \tag{3.9}$$

where x is the input data. $\text{SubLayer}(x)$ is the result of the computation performed by the sub-layer. $x + \text{SubLayer}(x)$ is the residual connection, adding the input to the output. $\text{LayerNorm}(\cdot)$ is a normalization operation used to stabilize the output.

(3) Feed-forward neural network:After the attention mechanism fuses information, the feed-forward network performs feature enhancement to extract deeper and more abstract representations. In the standard Transformer encoder, the FFN is placed after the residual connection. It consists of two fully connected layers: the first projects each position's representation into a higher-dimensional space, and the second maps it back to the original dimension. This structure allows the FFN to introduce non-linear transformations, enhancing the model's expressive power. Additionally, since it processes each position independently, it effectively captures complex patterns in the sequence.

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{3.10}$$

where$x$ is the input representation. W_1, W_2 are the weight matrices for the two linear transformations. b_1, b_2 are the bias terms. The activation function is generally ReLU (i.e., $\max(0, x)$).

3.4.2.4 Global Average Pooling

In natural language processing tasks, the input and output sequences typically represent two semantically different sequences, and the output needs to be generated token by token—for example, in tasks such as machine translation, text generation, and question answering, which are common applications of large language models. In these tasks, both the input and the output are natural language sequences, and the decoder serves as a "generator" that is primarily responsible for producing the output sequence. However, in time series forecasting tasks, the input and output sequences have the same semantics, with the only difference being a shift in time. Therefore, there is no need to generate discrete tokens, no need for step-by-step autoregressive prediction, and no need for target embeddings—hence, the decoder is not required. In this study, we replace the decoder structure with pooling and fully connected layers to produce the final output.

The output of the Transformer layer is followed by a pooling layer, whose primary purpose is to reduce the dimensionality of the high-dimensional tensor produced by the encoder. Specifically, the input to the pooling layer has the shape (batch_size, maxlen, embed_dim), where maxlen corresponds to the length of the historical input sequence and embed_dim is set to 64. After compression, the resulting shape becomes (batch_size, embed_dim). In this process, the pooling layer compresses the sequence length dimension, aggregating information from all historical time steps into a single global context vector, while preserving the feature dimension. This approach is based on the fact that the feature dimension captures the core abstract characteristics of the sequence, with each dimension encoding specific patterns such as periodicity, trends, or fluctuations. Compressing this dimension would result in a complete loss of expressive power. The pooling layer thus enables global representation extraction, reduces the computational burden of the model, and provides more stable output.

3.4.2.5 Fully Connected Layer

This fully connected layer further increases the feature dimension by mapping the global context vector to a higher-dimensional hidden space, thereby providing richer feature support for the subsequent output layer. This design helps the model fully integrate and reorganize historical information before generating prediction values. Specifically, the feature dimension is expanded from 64 to 1024 through this dense

layer.

3.4.2.6 Output Layer

The output layer is the final layer of the model and directly produces the final prediction results. It maps a 1024-dimensional feature vector into a prediction sequence of length equal to that of the target sequence Y. Specifically, the output layer takes the tensor of shape $(\text{batch\_size},\ 1024)$ generated by the previous fully connected layer and directly projects it into the task output of shape $(\text{batch\_size},\ \text{maxlen})$, where maxlen represents the length of the target sequence Y. In our design, maxlen is set to be the same as the length of the historical input sequence X.

3.4.3 Hyperparameter settings

3.4.3.1 Loss Function

The loss function serves precisely to "guide and correct the model to generate predictions that better match the actual results." In time series forecasting, the loss function compares the model's predicted value at each time step with the corresponding ground truth value and computes the error between them. These individual errors are then propagated backward through the network via the backpropagation mechanism, guiding the update of the model's parameters. In this way, the model can adjust its internal weights based on the prediction deviation at each time step. Through multiple training iterations, the model gradually improves, and its predictions increasingly align with the true sequence at both the global and local levels.

This study adopts the Mean Squared Error (MSE) loss, which is the most commonly used loss function in time series forecasting tasks.The calculation formula for MSE is:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y_{\text{true},i} - y_{\text{pred},i}\right)^2 \tag{3.11}$$

Where $y_{\text{true},i}$ is the true value, $y_{\text{pred},i}$ is the predicted value, and n is the number of samples. The smaller the MSE value, the smaller the difference between the model's predicted results and the true values.

3.4.3.2 Activation Function

The activation function is responsible for the output of neurons and determines the activation state (or output) of each layer in the neural network. Activation functions are typically non-linear (such as ReLU, Sigmoid, Tanh, etc.), enabling neural networks to learn and express complex non-linear relationships. The activation function influences the output of each layer, thereby affecting the gradient calculations of subsequent layers. During backpropagation, the gradient of the activation function needs to be computed so that the parameters can be updated during gradient descent.

In this study, the activation function used is ReLU, which is the default activation function in the standard Transformer architecture [69]. ReLU is applied in both the Transformer layers and other layers of the model. As a nonlinear activation function, ReLU sets negative values to zero while keeping positive values unchanged. This effectively enhances the representational power of the neural network and offers high computational efficiency. Its main advantage lies in avoiding the vanishing gradient problem,

thereby enabling more effective training of deep networks.The formula for ReLU is:

$$\text{ReLU}(x) = \max(0, x) \tag{3.12}$$

3.4.3.3 Optimizer

The optimizer adjusts the model's parameters based on the error from the loss function through gradient descent (or other optimization algorithms). It uses the gradient information calculated by backpropagation to update the model's weights and biases according to a specific rule, with the goal of minimizing the loss function. After the gradients for each layer are computed via backpropagation, the optimizer uses these gradients to update the parameters. Optimizers like SGD, Adam, RMSprop, etc., use different strategies to update parameters, such as adjusting the step size through the learning rate.

Similarly, this study adopts Adam (Adaptive Moment Estimation), the default optimizer used in the standard Transformer architecture [69]. Adam is a gradient-based optimization algorithm that combines the concepts of momentum and adaptive learning rates, allowing it to automatically adjust the learning rate for each parameter, thereby improving training efficiency. The Adam optimizer updates parameters using the following formula:

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{v_t} + \epsilon} m_t \tag{3.13}$$

Where θ_t represents the parameters, m_t and v_t are the first and second moment estimates of the gradients, η is the learning rate, and ϵ is a small value added to avoid division by zero. Adam can effectively handle sparse gradients and large datasets.

3.4.3.4 Regularization

Regularization techniques aim to reduce overfitting in models, helping them generalize better on complex datasets. Common regularization methods include L1, L2 regularization, and Dropout. Regularization adds a penalty term to the loss function, limiting the excessive changes in model parameters. This penalty term affects the calculation of the loss function, so during backpropagation, the impact of the penalty term is also considered, which restricts the parameter updates and prevents overfitting.

The standard Transformer [69]applies Dropout as its default regularization method, with a typical rate of 0.1, and integrates it into multiple parts of the architecture including attention layers, feed-forward networks, and residual connections, to mitigate overfitting and improve generalization.These standard practices were used in this study.

3.4.3.5 Learning rate

The learning rate controls the step size of each parameter update. Although the standard Transformer introduces a custom learning rate scheduling formula, in most engineering practices or model variants, fixed learning rates or simplified scheduling strategies are more commonly used. A fixed learning rate of 1e-3 is the most widely adopted setting, especially in time series forecasting tasks. In this study, we adopt a learning rate of 1e-3.

3.4.3.6 Batch size

Set to 64, meaning that 64 samples are used in each training iteration.In this study, taking the US Air dataset as an example, the training set has a total length of 9000. Each sample consists of a historical sequence (input) of length 1000 and a target sequence (output) of length 1000. We adopt a sliding window strategy to construct training samples, with the window moving forward by one time step at each iteration. Since each sample occupies a total of 2000 time steps, the number of valid starting positions is calculated as $9000 - 1000 - 1000 + 1 = 7001$. As a result, a total of 7001 valid samples can be generated from the training set for model training.

3.4.3.7 Epochs

The total number of training epochs is set to 50, meaning the entire training set will be used for 50 training iterations.

3.4.3.8 Hyperparameters in the Transformer block

(1) Embedding dimension: In most Transformer-based time series forecasting tasks, the embedding dimension is commonly set to 64 or 128, it is set to 64.

(2) Number of heads: In most Transformer-based time series forecasting tasks, the number of attention heads is typically set to 4 or 8,it is set to 4.

(3) Maximum sequence length : Set to 1000 for USAir, 500 for arXiv data, and 400 for PGP.

3.4.4 Prediction result evaluation

The most intuitive way to evaluate prediction results is to visualize them and directly compare the differences between the predicted trajectory and the real trajectory. In addition, we also use several common prediction metrics in time series forecasting to provide a more comprehensive and detailed evaluation. We evaluated the prediction performance from multiple aspects, including the average error accuracy between the predicted and real values, trend matching, and detail matching. The following are the evaluation metrics used in this research.

3.4.4.1 R²

R² (Coefficient of Determination) is a common metric to assess how well the predicted values approximate the actual data. It indicates the proportion of the variance in the true values that is predictable from the model. R² ranges from negative infinity to 1, where values closer to 1 indicate better performance, and negative values suggest the model performs worse than a simple mean predictor.

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2} \tag{3.14}$$

Where y_i represents the i-th ground truth value, and $\hat{y}_i$ represents the i-th predicted value. $\bar{y}$ denotes the mean of the ground truth values. The term $\sum(y_i - \hat{y}_i)^2$ is the Sum of Squared Errors , which measures

the total prediction error between the model's outputs and the actual values. The term $\sum(y_i - \bar{y})^2$ is the Total Sum of Squares , which represents the variance of the ground truth values relative to their mean.

3.4.4.2 Mean Absolute Relative Error (MARE)

MARE quantifies the average magnitude of relative errors between predicted and actual values. It expresses prediction errors as a proportion of the true values, making it suitable for evaluating accuracy across varying scales. Lower MARE values indicate better predictive performance.

$$\text{MARE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{3.15}$$

Where y_i represents the i-th ground truth value, and $\hat{y}_i$ represents the i-th predicted value. n is the number of samples . The term $\left|\frac{y_i - \hat{y}_i}{y_i}\right|$ represents the relative error of the i-th prediction with respect to the ground truth value.

3.4.4.3 Mean Bias Deviation (MBD)

MBD indicates the overall systematic bias of the model. It is the average of all prediction errors. A positive MBD implies the model tends to overestimate, while a negative MBD suggests underestimation. Smaller absolute values of MBD represent a more unbiased and reliable model.

$$\text{MBD} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i) \tag{3.16}$$

Where $\hat{y}_i$ represents the i-th predicted value, y_i is the i-th ground truth value, and n denotes the number of samples. The term $\hat{y}_i - y_i$ indicates the prediction error for the i-th sample, where a positive value means overestimation and a negative value means underestimation.

3.4.4.4 Correlation

Correlation measures the linear relationship between the predicted sequence and the true sequence. It is typically quantified using the Pearson correlation coefficient, which ranges from -1 to 1. A value close to 1 indicates strong alignment in trends between the two sequences, a value close to -1 indicates completely opposite trends, and a value near 0 implies no linear correlation.

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2} \cdot \sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{3.17}$$

Where x_i and y_i represent the values of variables x and y for the i-th sample, respectively. $\bar{x}$ and $\bar{y}$ are the mean values of variables x and y. The numerator is the covariance, which reflects the degree to which the two variables change together. The denominator is the product of the standard deviations of the two variables, serving as a normalization factor.

3.4.4.5 Dynamic Time Warping (DTW)

DTW is a distance metric that evaluates the similarity between two time series by aligning them in time. It allows stretching or compressing of the time axis to find the best match, making it especially useful when the sequences differ in length or exhibit temporal misalignment. A smaller DTW value indicates higher similarity.

$$D(i,j) = d(x_i, y_j) + \min \begin{cases} D(i-1,j) \\ D(i,j-1) \\ D(i-1,j-1) \end{cases} \tag{3.18}$$

where $D(i,j)$ represents the minimum accumulated distance to the point (i,j); $d(x_i, y_j)$ is the local distance between two points, commonly calculated using Euclidean distance:

$$d(x_i, y_j) = (x_i - y_j)^2 \quad \text{or} \quad |x_i - y_j| \tag{3.19}$$

Boundary condition: $D(1,1) = d(x_1, y_1)$, and the rest are initialized to $+\infty$. The final DTW distance is $D(n,m)D(n,m)$, which represents the minimum total matching cost between the two entire sequences.

3.4.4.6 Sequence Consistency

Sequence consistency evaluates whether the predicted sequence captures the local variation patterns of the true sequence. It is computed by analyzing the correlation of the differences between adjacent time steps (i.e., first-order differences), thus reflecting how well the model captures the dynamic behaviors of the data.

$$\text{Sequence Consistency} = \frac{\sum_{i=1}^{n-1} (\Delta y_i - \Delta\bar{y})\,(\Delta\hat{y}_i - \Delta\hat{y})}{\sqrt{\sum_{i=1}^{n-1} (\Delta y_i - \Delta\bar{y})^2} \cdot \sqrt{\sum_{i=1}^{n-1} (\Delta\hat{y}_i - \Delta\hat{y})^2}} \tag{3.20}$$

where $\Delta y_i = y_{i+1} - y_i$ and $\Delta\hat{y}_i = \hat{y}_{i+1} - \hat{y}_i$ represent the i-th changes in the ground truth and predicted sequences, respectively. $\overline{\Delta y}$ and $\overline{\Delta\hat{y}}$ denote the mean values of these differenced sequences. The numerator captures the covariance between the two sequences, while the denominator is the product of their standard deviations.

Among them, MARE and MBD evaluate the prediction performance based on the precision difference between the predicted values and the real values. Correlation, on the other hand, evaluates the performance from the perspective of trend consistency. However, it is more sensitive to numerical discrepancies, and deviations in the details will directly lower its value. In contrast, Correlation has a higher tolerance for both numerical and detail discrepancies, focusing more on the matching of trends over a long time scale. DTW measures the overall similarity, allowing for time misalignment in details, and its primary focus is on the shape similarity. Sequence Consistency combines both trend and detail matching, making it a comprehensive metric for evaluating time series prediction.

4 Experimental Results

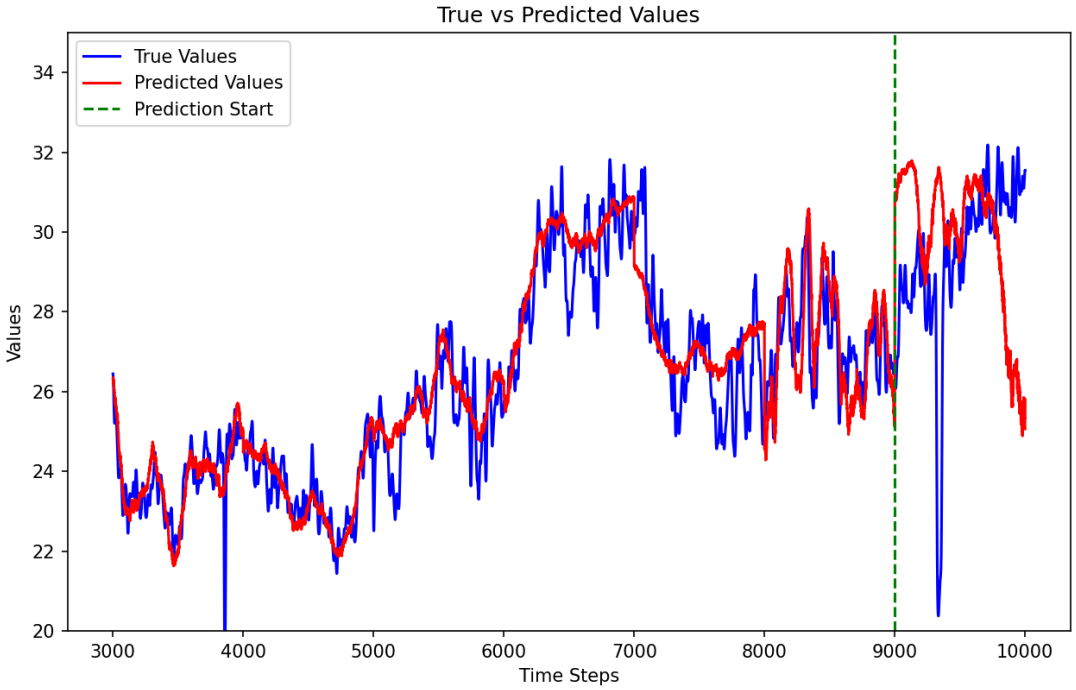

Figure 4.1: USAir, the node BIL, The predicted(red) and real(blue) trajectories of popularity.(The green dashed line separates the training process (before) from the testing process (after), with subsequent results being the same.).

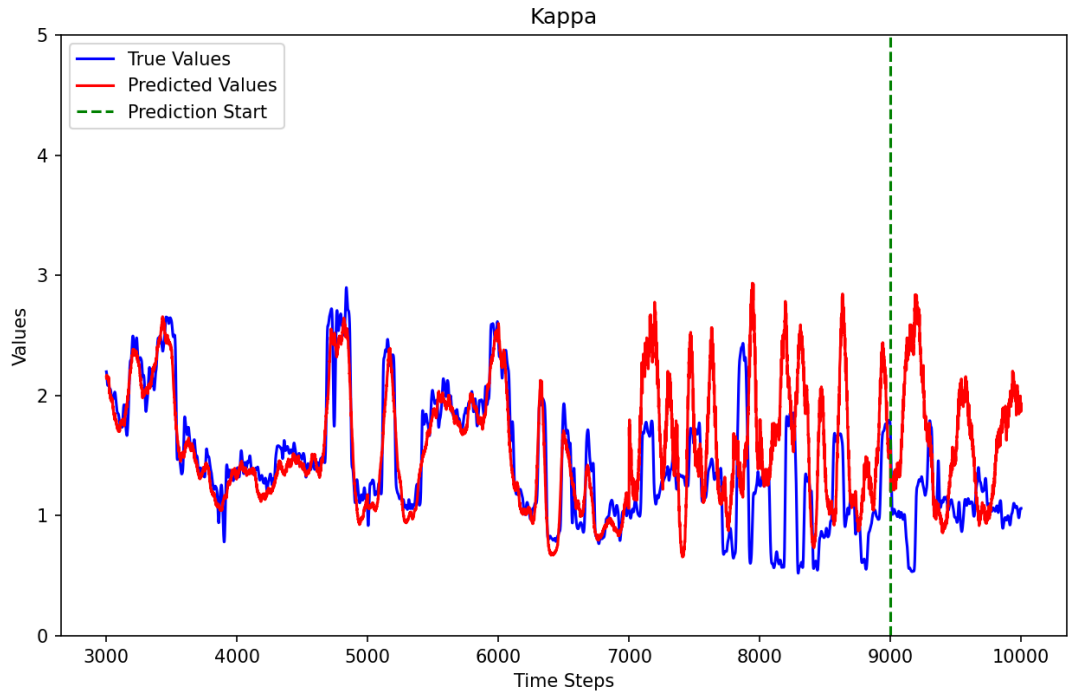

Figure 4.2: USAir, the node BIL, The predicted(red) and real(blue) trajectories of expected degree.

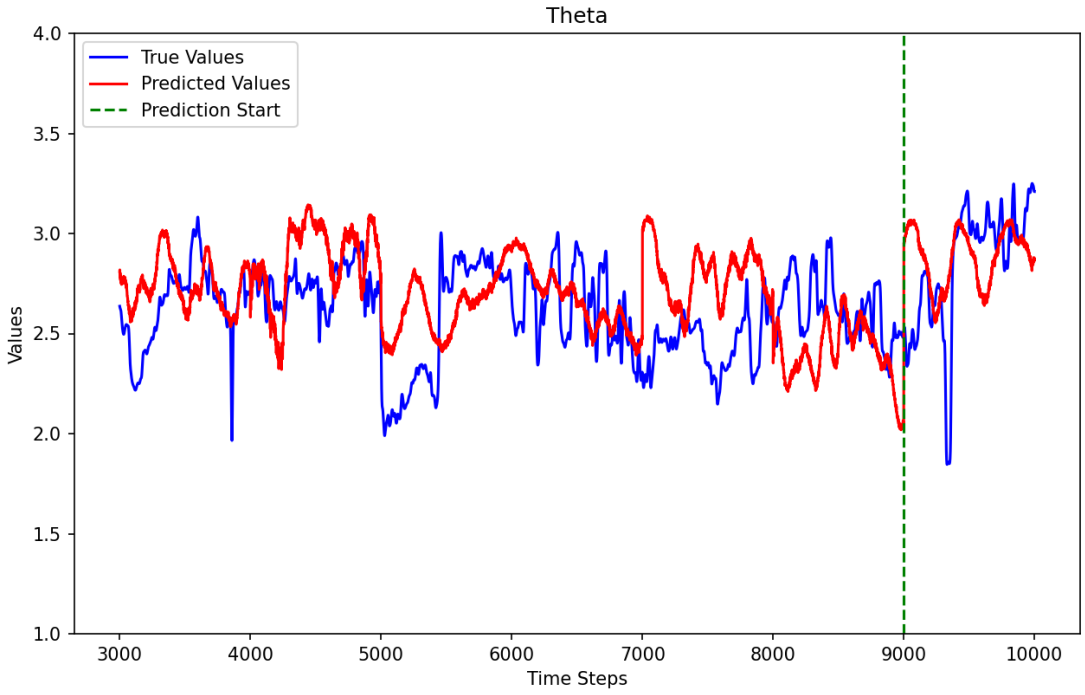

Figure 4.3: USAir, the node BIL, The predicted(red) and real(blue) trajectories of similarity.

Table 4.1: USAir, the node BIL

	R^2	MARE	MBD	Correlation	DTW	Sequence Consistency
popularity						
train	0.9213	0.0247	0.0761	0.962	3.3e-07	0.0381
test	-1.497	0.087	0.525	-0.421	1.8e-07	-0.018
kappa						
train	0.881	0.129	-0.072	0.944	2.1e-09	0.091
test	-56.39	1.144	0.927	-0.002	8.5e-07	0.011
similarity						
train	-1.180	0.0998	0.0147	0.251	8.7e-09	0.0108
test	-0.057	0.09179	0.043	0.180	1.0e-07	0.0208

The prediction results for node BIL in the USAir network indicate that the time series is highly unpredictable. In terms of point-wise similarity, both *popularity* and *kappa* achieve strong performance on the training set, with R^2 values of 0.9213 and 0.881, respectively, indicating excellent model fitting. However, their performance drops drastically on the test set, with R^2 values falling to -1.497 and -56.39, respectively, reflecting a complete failure in generalization. Although *similarity* shows low R^2 in both sets, it remains relatively stable (-1.180 train, -0.057 test).

Regarding trend similarity, *popularity* and *kappa* exhibit high correlation in the training set (0.962 and 0.944), but these values collapse on the test set (-0.421 and -0.002), suggesting the model fails to capture or generalize trend direction. *Similarity* shows slightly higher test correlation (0.180), though still weak.

For shape similarity, while DTW values remain small across all features, indicating global shape alignment, the Sequence Consistency scores are generally low on the test set. In particular, *popularity* shows a negative value (-0.018), indicating disordered local trend directions.

Overall, while the model performs well during training, its test performance collapses across all metrics. Node BIL's time series exhibits strong non-stationarity, trend instability, and structural unpredictability, rendering accurate forecasting highly challenging.

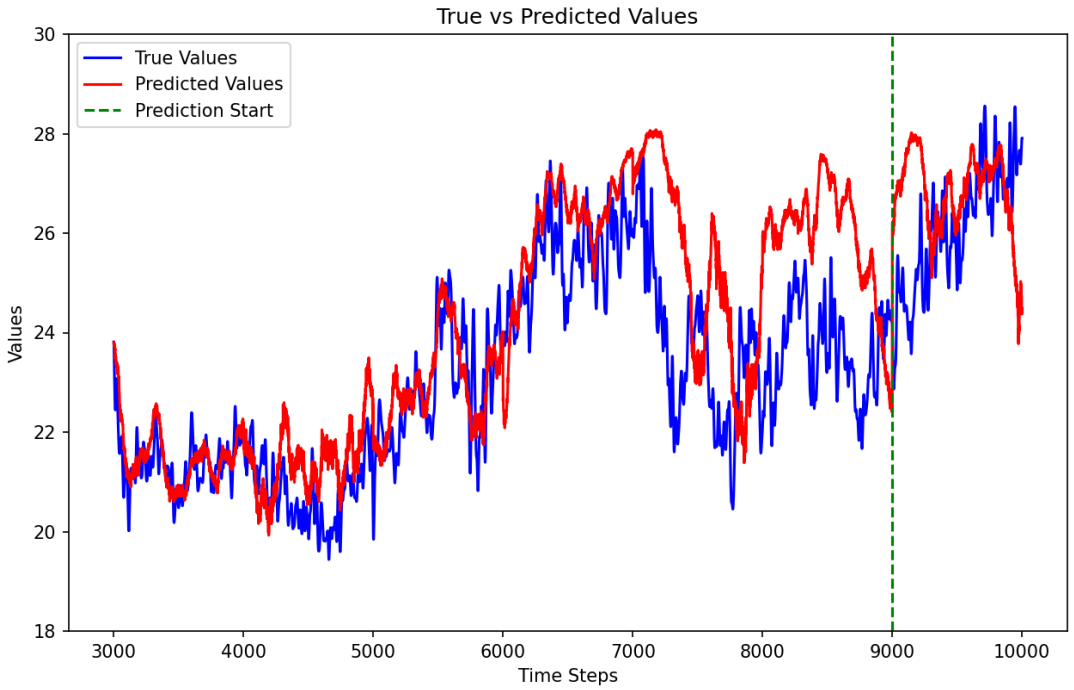

Figure 4.4: USAir, the node BUR, The predicted(red) and real(blue) trajectories of popularity.

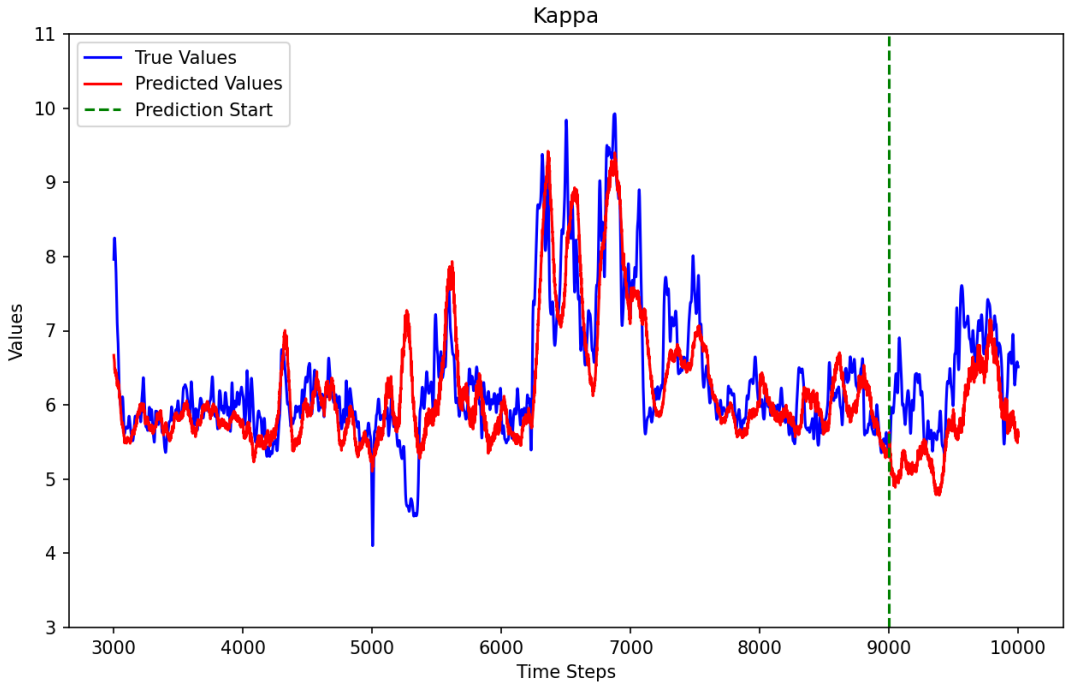

Figure 4.5: USAir, the node BUR, The predicted(red) and real(blue) trajectories of expected degree.

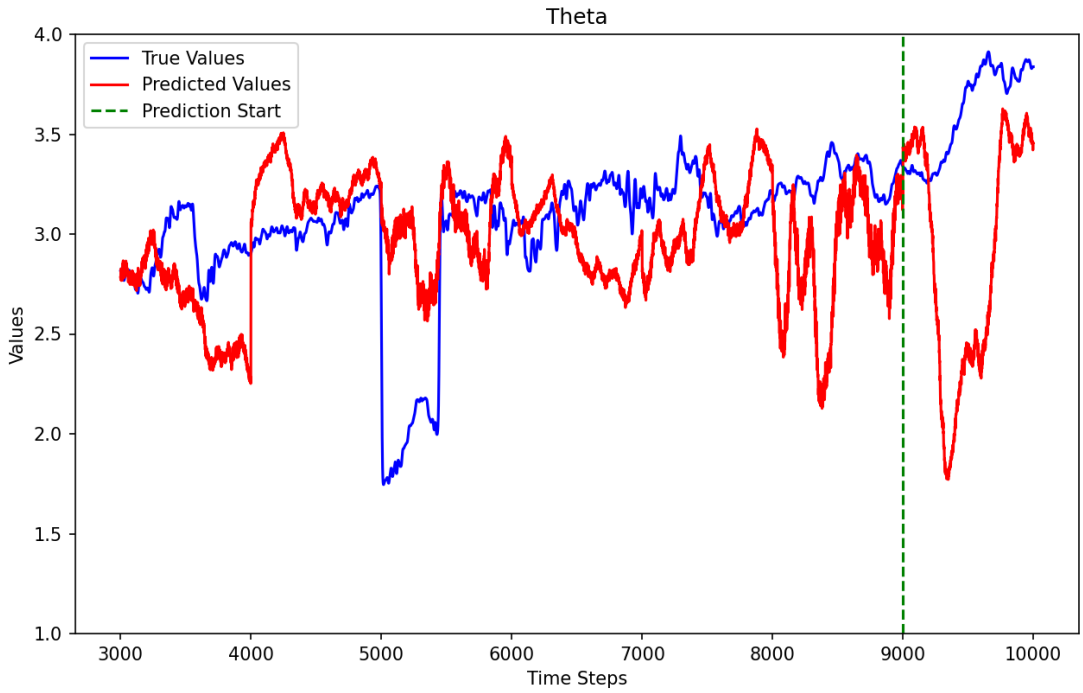

Figure 4.6: USAir, the node BUR, The predicted(red) and real(blue) trajectories of similarity.

Table 4.2: USAir, the node BUR

	R^2	MARE	MBD	Correlation	DTW	Sequence Consistency
popularity						
train	0.594	0.0464	0.637	0.893	1.8e-08	0.0191
test	-1.111	0.054	0.712	-0.244	1.5e-06	-0.002
kappa						
train	0.744	0.058	-0.112	0.874	2.5e-08	0.050
test	-0.857	0.106	-0.667	0.723	2.9e-07	-0.015
similarity						
train	-0.114	0.117	0.017	0.339	7.8e-09	0.022
test	-15.29	0.2071	0.043	0.0508	9.7e-07	0.004

The prediction results for the BUR node indicate that the time series exhibits very low predictability. In terms of point-wise similarity, the R^2 values on the test set are all negative, with particularly poor results for the *similarity* feature ($R^2 = -15.29$), indicating that the model fails to approximate future values accurately. Regarding trend similarity, while *popularity* and *similarity* show weak or even negative correlation on the test set, *kappa* is the only feature that retains a moderate level of correlation (0.723), suggesting partial preservation of trend direction. For shape similarity, although DTW values remain relatively low, indicating some global shape alignment, the sequence consistency values are close to zero or negative, reflecting disordered or reversed local trends. Overall, the prediction performance on the BUR node reveals high uncertainty and structural variability in the time series, with only *kappa* providing

marginally usable predictive information.

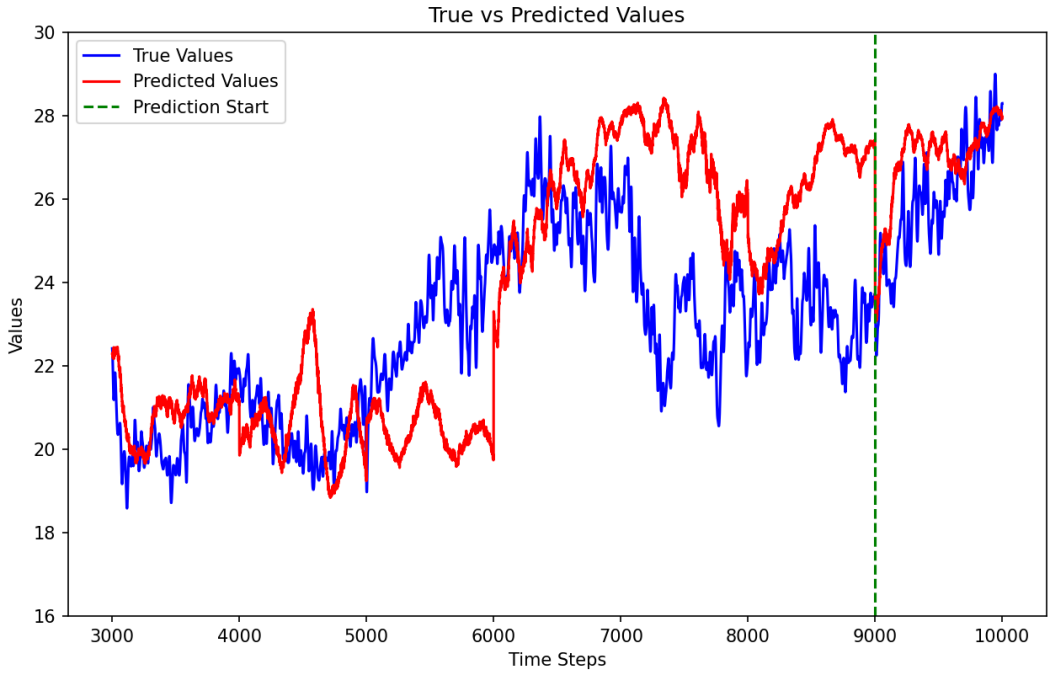

Figure 4.7: USAir, the node COS, The predicted(red) and real(blue) trajectories of popularity.

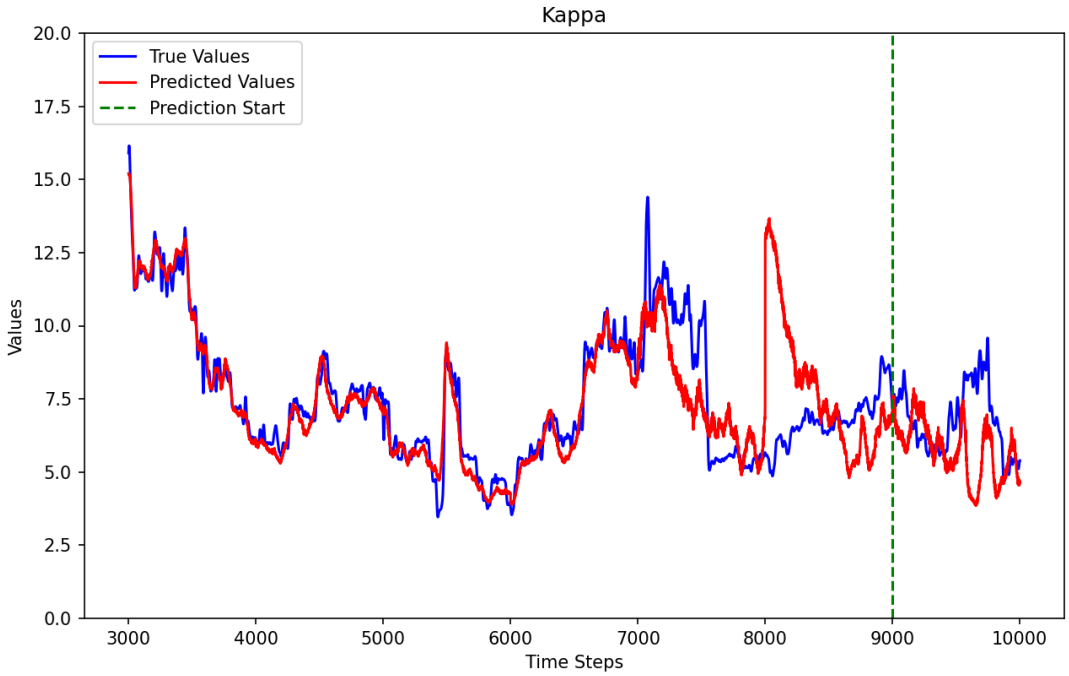

Figure 4.8: USAir, the node COS, The predicted(red) and real(blue) trajectories of expected degree.

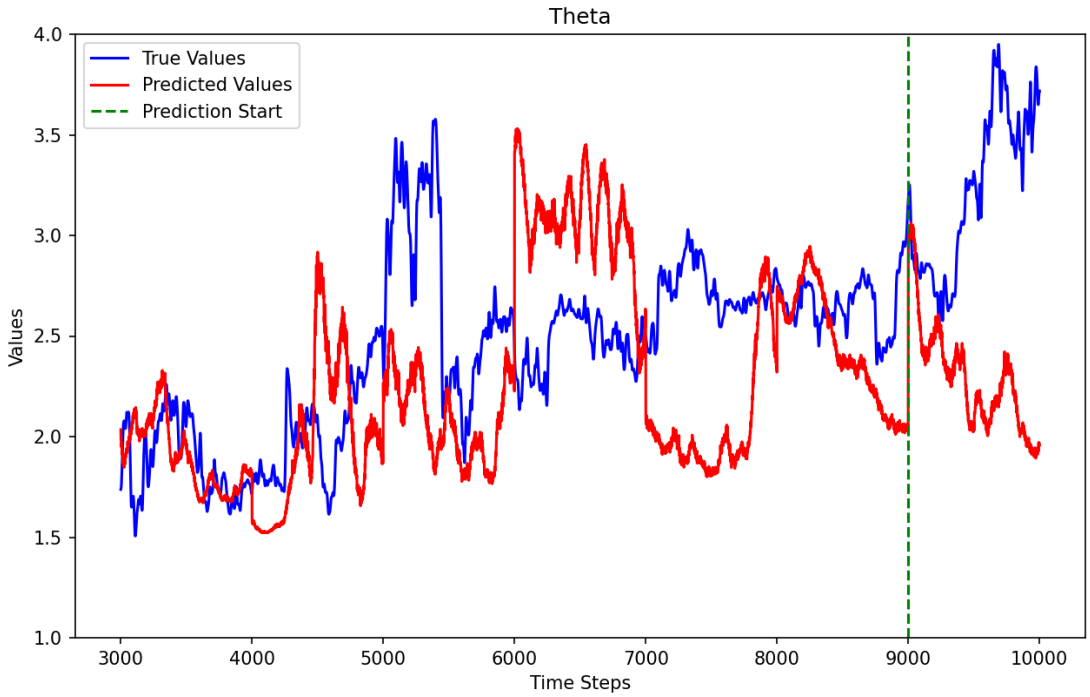

Figure 4.9: USAir, the node COS, The predicted(red) and real(blue) trajectories of similarity.

Table 4.3: USAir, the node COS

	R^2	MARE	MBD	Correlation	DTW	Sequence Consistency
popularity						
train	-0.038	0.072	0.611	0.722	2.2e-07	0.013
test	0.006	0.043	0.912	0.676	1.1e-06	0.016
kappa						
train	0.694	0.108	-0.040	0.8503	9.2e-09	0.082
test	-1.61	0.186	-1.023	-0.0571	9.8e-07	0.011
similarity						
train	-0.66	0.181	-0.077	0.374	1.5e-09	0.207
test	-6.789	0.2741	-0.931	-0.595	5.2e-06	-0.006

The prediction results for node COS in the USAir network indicate generally low predictability, particularly on the test set. In terms of point-wise similarity, while *popularity* and *kappa* achieve moderately acceptable performance on the training set (with R^2 values of -0.038 and 0.694), both *kappa* and *similarity* exhibit significantly negative R^2 values on the test set (-1.61 and -6.789), suggesting complete failure in predicting future numerical values.

Regarding trend similarity, the correlation values on the test set are notably weak. *Kappa* and *similarity* exhibit degraded or reversed trends with correlations of -0.0571 and -0.595, respectively. Only *popularity* maintains a moderate correlation of 0.676, indicating some level of stability in directional patterns.

In terms of shape similarity, DTW values remain relatively low, implying that the predicted sequences are roughly aligned in shape. However, the sequence consistency values on the test set are generally low, particularly for *similarity*, which drops to a negative value (-0.006), revealing highly disordered local trend directions.

Overall, the time series of node COS shows large numerical deviations, inconsistent trends, and weak structural stability in the test set, with only *popularity* preserving limited predictive ability in terms of trend consistency.

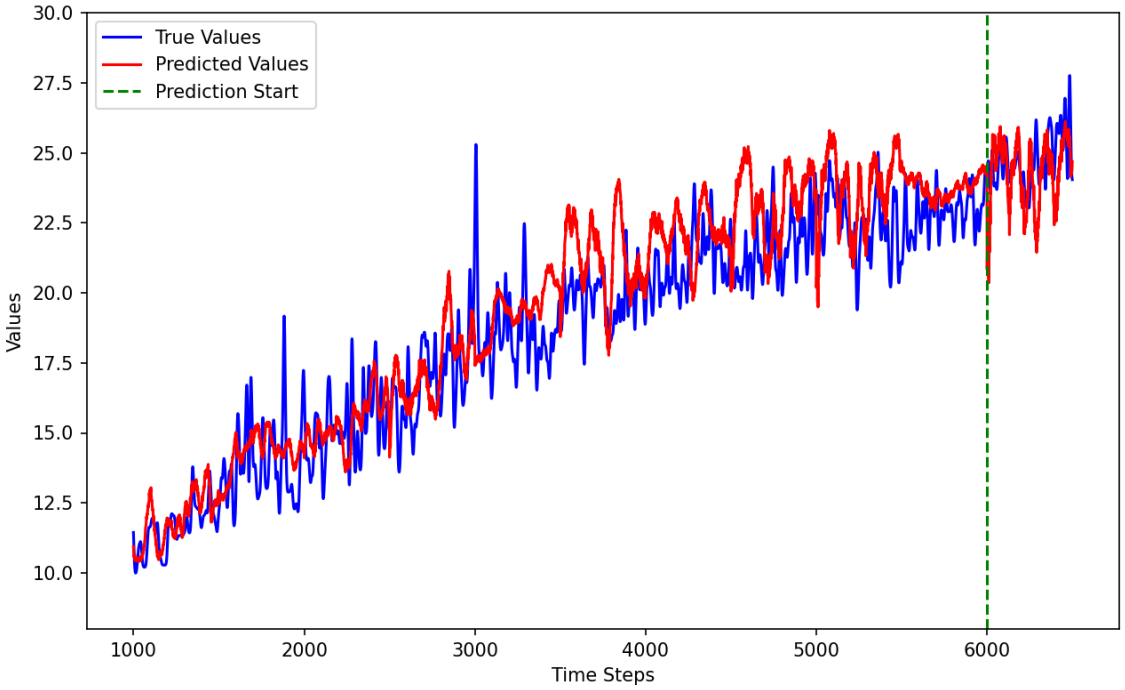

Figure 4.10: arXiv, the node 10, The predicted(red) and real(blue) trajectories of popularity.

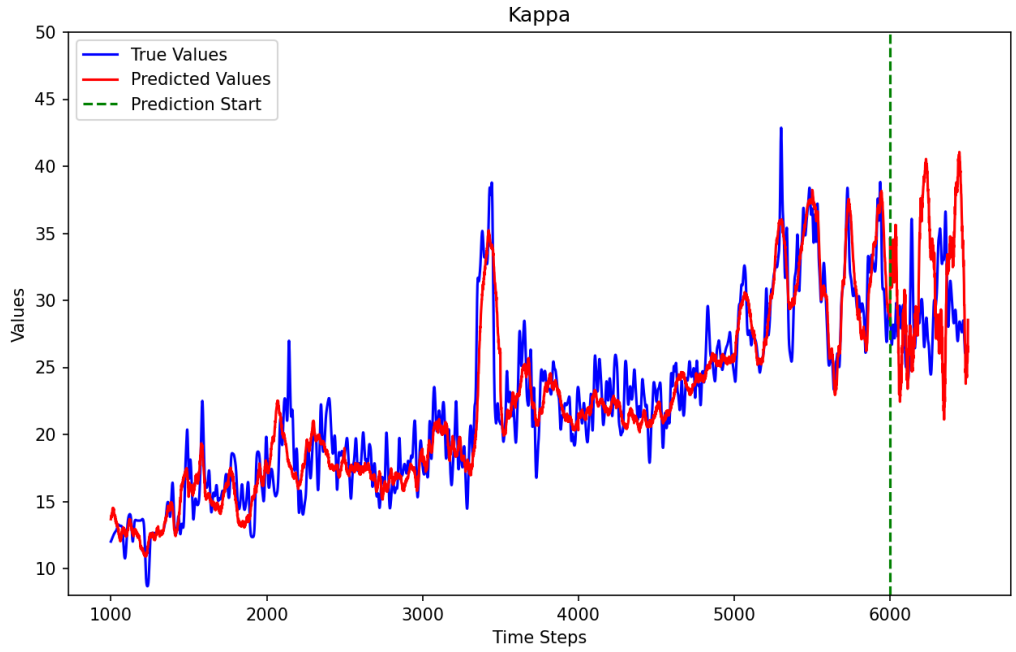

Figure 4.11: arXiv, the node 10, The predicted(red) and real(blue) trajectories of expected degree.

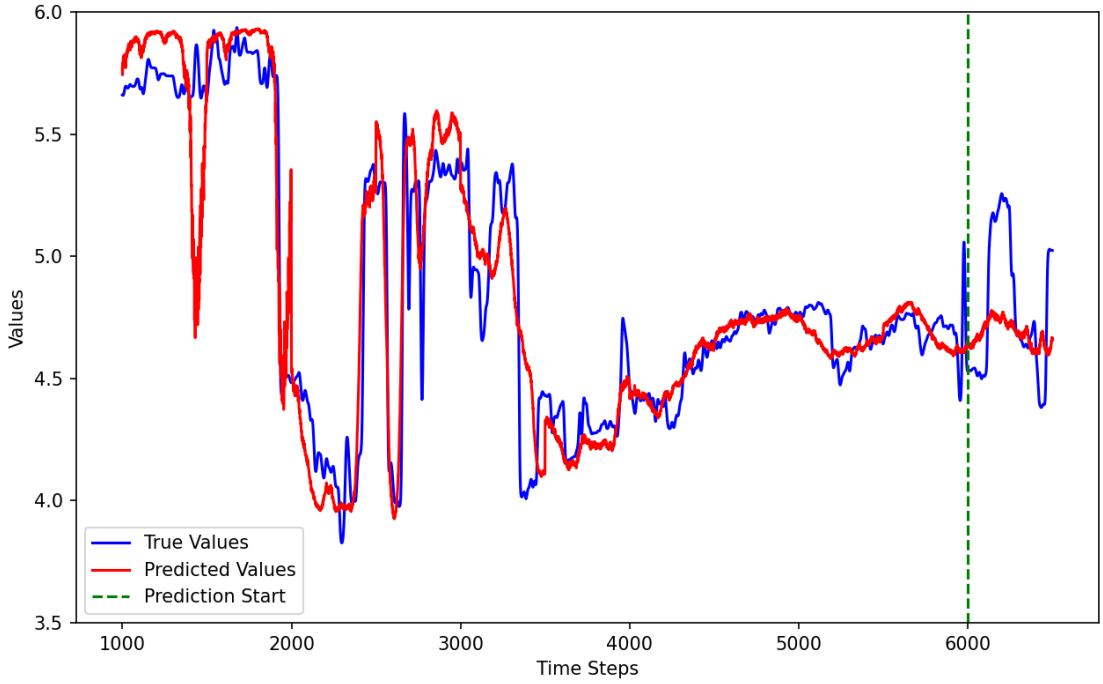

Figure 4.12: arXiv, the node 10, The predicted(red) and real(blue) trajectories of similarity (c).

Table 4.4: arXiv, the node 10

	R^2	MARE	MBD	Correlation	DTW	Sequence Consistency
popularity						
train	0.807	0.072	0.835	0.937	3.9e-08	0.005
test	-1.275	0.0414	-0.453	0.255	1.1e-05	0.0551
kappa						
train	0.895	0.0731	-0.228	0.947	6.3e-08	0.127
test	-5.201	0.1937	2.396	-0.263	5.4e-07	0.024
similarity						
train	0.876	0.026	0.007	0.942	2.2e-08	0.085
test	0.0781	0.0445	-0.095	0.636	1.6e-06	0.011

The prediction results for node 10 in the arXiv network show good performance on the training set, but a significant degradation on the test set. In terms of point-wise similarity, all three features—*popularity*, *kappa*, and *similarity*—achieve high R^2 values on the training set (0.807, 0.895, and 0.876, respectively). However, in the test set, both *popularity* and *kappa* experience large drops in R^2 to negative values (-1.275 and -5.201), indicating a failure to generalize. In contrast, *similarity* retains a small but positive R^2 of 0.0781, suggesting limited predictive power.

With respect to trend similarity, *similarity* maintains a relatively high test correlation of 0.636, while *popularity* and *kappa* drop to 0.255 and -0.263, respectively, the latter indicating a reversed trend.

In terms of shape similarity, all features preserve low DTW values, suggesting global shape alignment. However, the Sequence Consistency metrics drop considerably on the test set, with *kappa* and *similarity* falling to 0.024 and 0.011, indicating unstable and disordered local trend structures.

Overall, although the model fits well on the training data, it suffers from poor generalization on unseen data. Only *similarity* demonstrates marginally usable predictive capacity in forecasting future values for node 10.

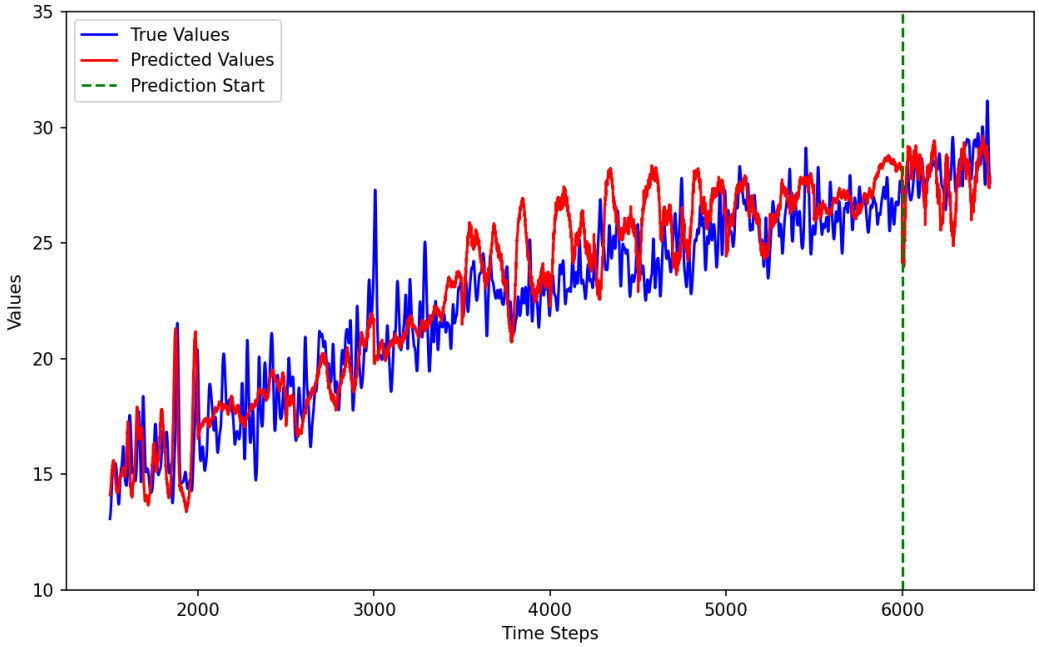

Figure 4.13: arXiv, the node 740, The predicted(red) and real(blue) trajectories of popularity.

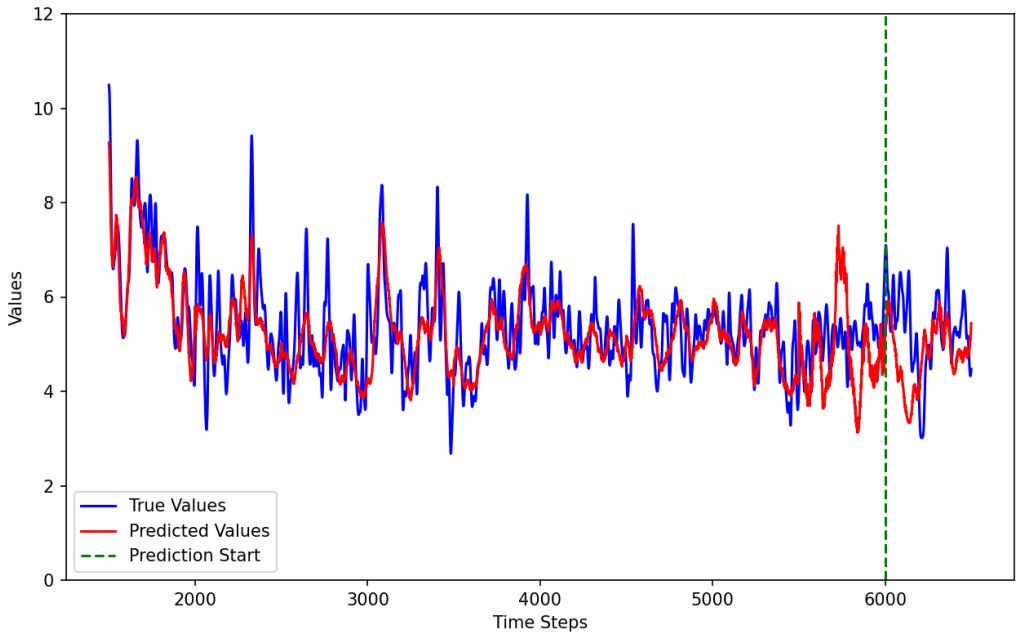

Figure 4.14: arXiv, the node 740, The predicted(red) and real(blue) trajectories of expected degree .

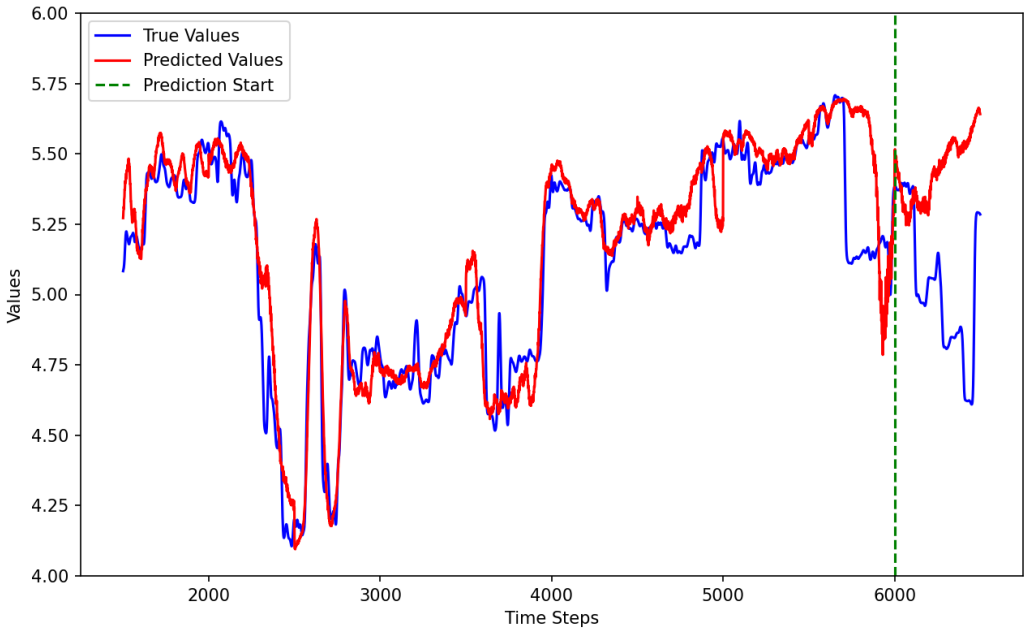

Figure 4.15: arXiv, the node 740, The predicted(red) and real(blue) trajectories of similarity.

Table 4.5: arXiv, the node 740

	R^2	MARE	MBD	Correlation	DTW	Sequence Consistency
popularity						
train	0.792	0.061	0.632	0.925	1.5e-06	0.018
test	-1.331	0.036	-0.436	0.253	3.9e-06	0.0516
kappa						
train	0.515	0.098	-0.135	0.743	1.3e-07	0.148
test	-0.732	0.168	-0.556	0.176	5.1e-06	0.036
similarity						
train	0.852	0.018	0.044	0.937	4.8e-08	0.121
test	-3.390	0.089	0.412	-0.470	8.3e-08	0.007

The prediction results for node 740 in the arXiv network indicate that the time series exhibits poor generalization performance. In terms of point-wise similarity, although all three features—*popularity*, *kappa*, and *similarity*—achieve relatively high R^2 scores on the training set (0.792, 0.515, and 0.852, respectively), their performance on the test set significantly deteriorates. The R^2 values for all features become negative, with *similarity* dropping sharply to -3.390, suggesting that the model fails to capture the actual dynamics of future values.

With respect to trend similarity, the test set correlation coefficients are considerably lower across all features. *Popularity* and *kappa* retain only weak positive correlations (0.253 and 0.176, respectively), while *similarity* demonstrates a clear trend reversal with a negative correlation of -0.470. This indicates

that the model not only struggles to follow the overall direction of the series but, in some cases, predicts opposing trends.

For shape similarity, the DTW values remain low in both the training and test sets, indicating that the overall sequence shapes are somewhat aligned. However, the Sequence Consistency values in the test set drop noticeably, especially for *similarity* (from 0.121 to 0.007), revealing that local trend directionality becomes unstable and inconsistent.

Overall, while the model fits the training data well, the generalization to unseen data is poor across all three features. The predictions on node 740's time series exhibit substantial numerical errors, disrupted trends, and degraded structural consistency, highlighting significant non-stationarity and structural volatility in the data.

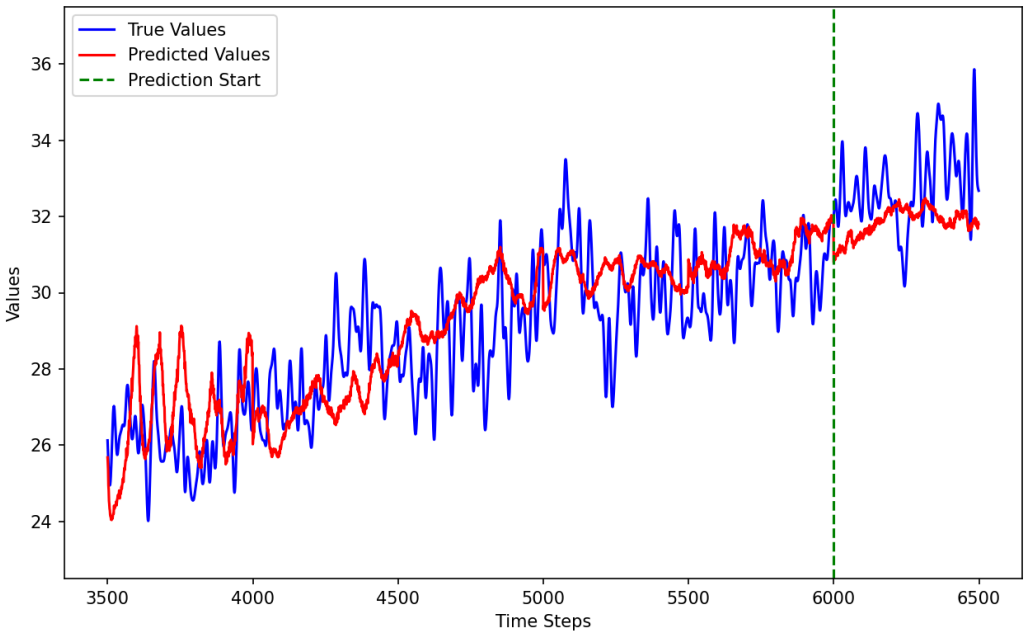

Figure 4.16: arXiv, the node 2247, The predicted(red) and real(blue) trajectories of popularity.

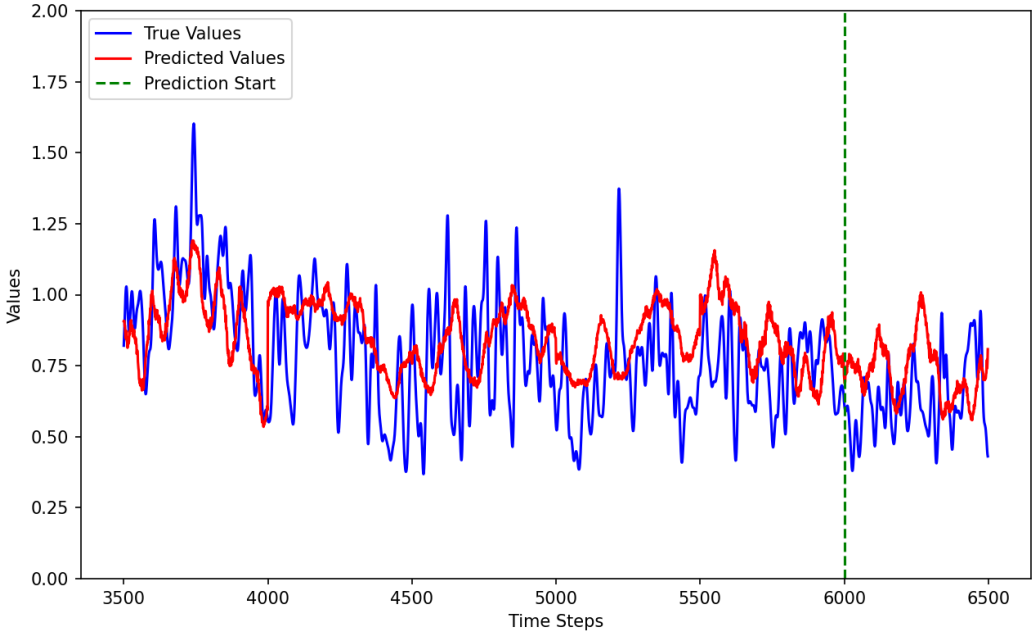

Figure 4.17: arXiv, the node 2247, The predicted(red) and real(blue) trajectories of expected degree.

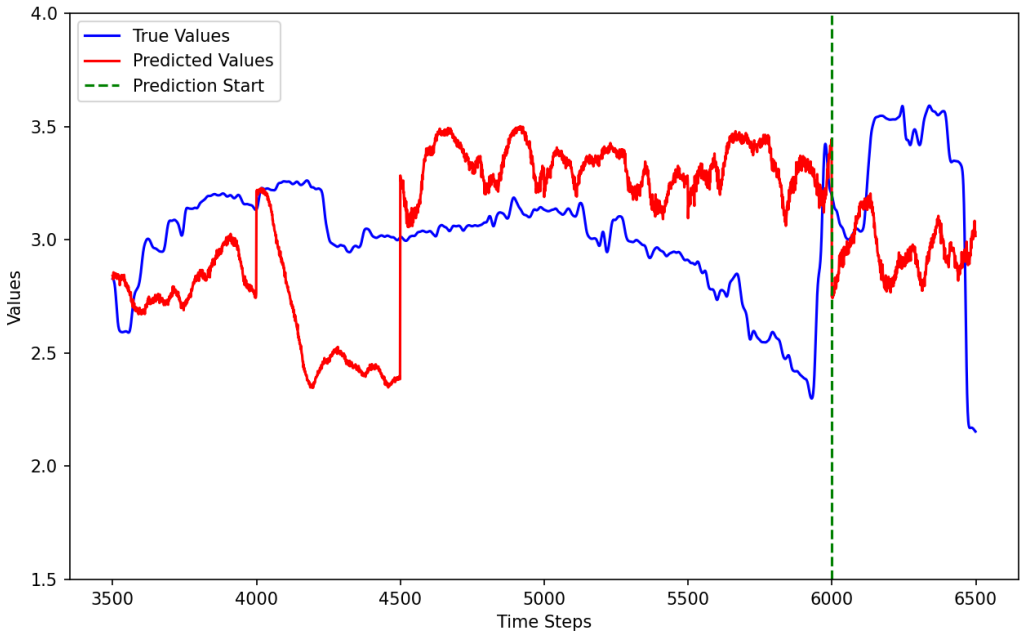

Figure 4.18: arXiv, the node 2247, The predicted(red) and real(blue) trajectories of similarity.

Table 4.6: arXiv, the node 2247

	R^2	MARE	MBD	Correlation	DTW	Sequence Consistency
popularity						
train	0.490	0.039	0.311	0.758	1.1e-07	0.014
test	-1.029	0.035	-0.954	-0.011	2.5e-06	0.024
kappa						
train	0.049	0.236	0.059	0.408	1.0e-08	0.008
test	-1.695	0.296	0.100	-0.265	1.6e-07	-0.036
similarity						
train	-3.510	0.135	0.085	-0.226	5.7e-08	-0.009
test	-1.147	0.141	-0.343	-0.203	8.7e-06	-0.010

The prediction results for node 2247 in the arXiv network reveal extremely low overall predictability, with poor model performance on both the training and test sets. In terms of point-wise similarity, all features exhibit negative R^2 values on the test set, indicating the model's inability to reconstruct future values. Notably, the *similarity* feature shows a highly negative R^2 of -3.510 even in the training set, suggesting an inherent lack of structure. While *popularity* achieves a moderate R^2 of 0.490 on the training set, it drops to -1.029 on the test set, indicating a complete failure in generalization.

With respect to trend similarity, the correlation of *popularity* drops from 0.758 (train) to -0.011 (test), showing an absence of consistent trend direction. Both *kappa* and *similarity* also exhibit low or negative correlation values on both sets, indicating that the model fails to capture any meaningful directional patterns.

For shape similarity, although DTW values remain relatively low on the training set, they increase noticeably on the test set. Sequence Consistency values are negative across all features on the test set, particularly for *kappa* and *similarity*, reflecting disordered and unstable local trend structures.

Overall, the time series of node 2247 lacks learnable structure in both training and test sets. The predictions suffer from large numerical deviations, reversed or inconsistent trends, and poor structural stability, making this node a typical case of strong non-stationarity and low predictability.

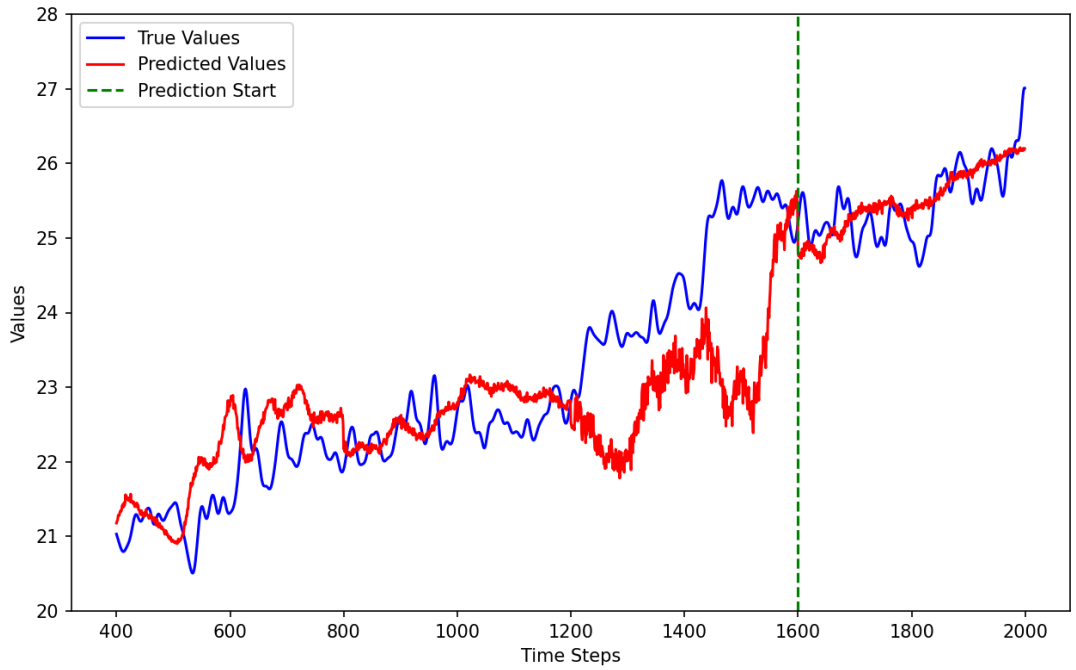

Figure 4.19: PGP, the node 0x0A0AC927, The predicted(red) and real(blue) trajectories of popularity.

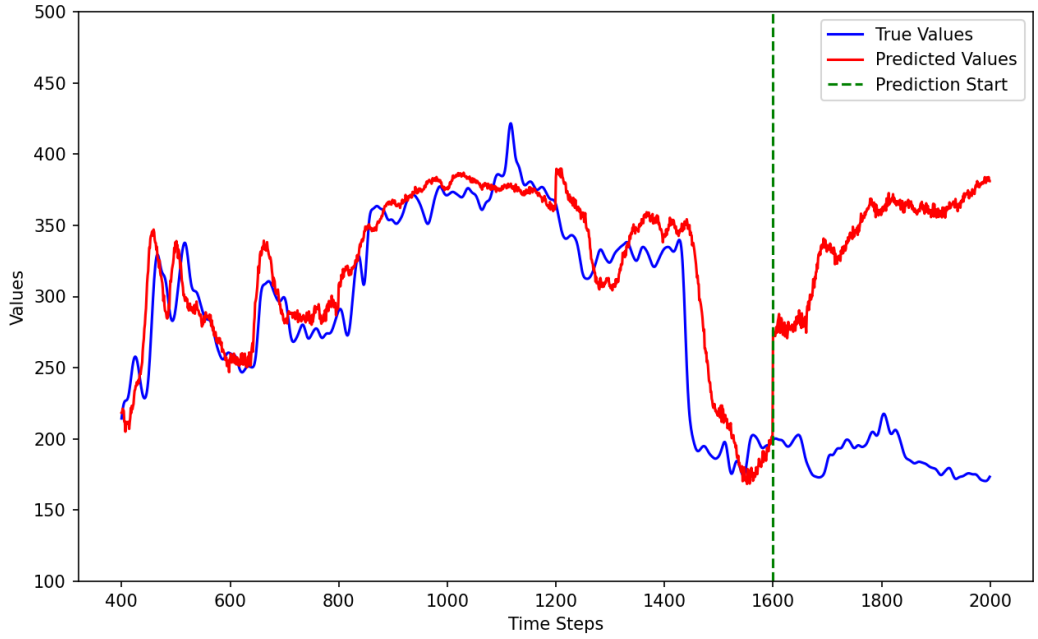

Figure 4.20: PGP, the node 0x0A0AC927, The predicted(red) and real(blue) trajectories of expected degree.

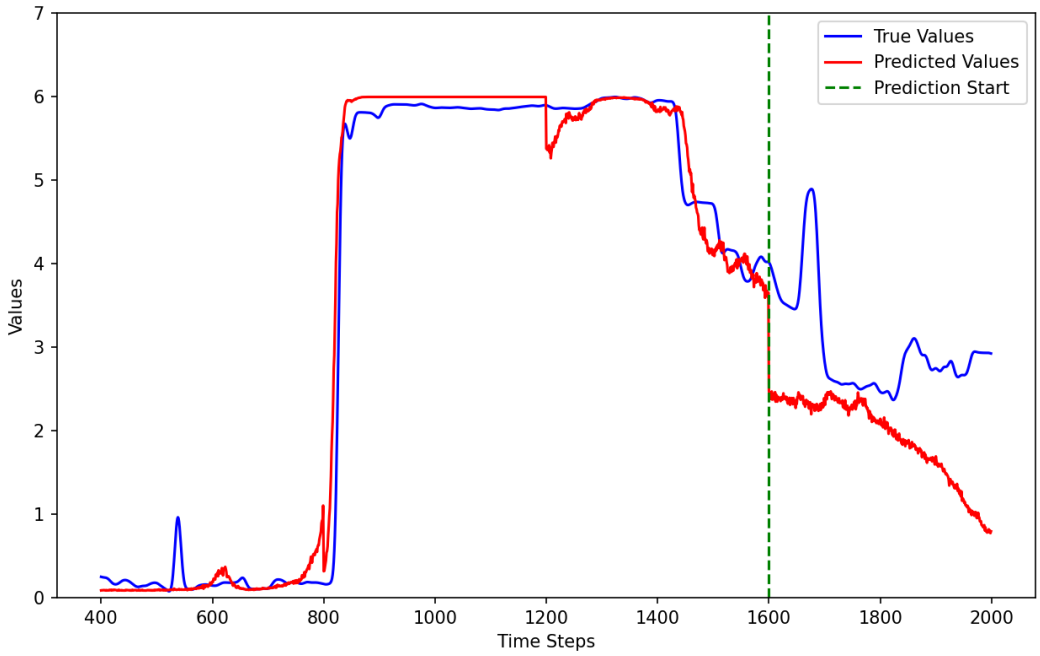

Figure 4.21: PGP, the node 0x0A0AC927, The predicted(red) and real(blue) trajectories of similarity.

Table 4.7: PGP, the node 0x0A0AC927

	R^2	MARE	MBD	Correlation	DTW	Sequence Consistency
popularity						
train	0.423	0.031	-0.247	0.682	5.5e-07	-0.023
test	0.409	0.011	0.050	0.678	1.1e-06	-0.021
kappa						
train	0.766	0.073	11.02	0.895	3.6e-05	0.056
test	-170.82	0.832	154.77	-0.286	53.05	-0.009
similarity						
train	0.375	0.265	0.048	0.991	2.9e-07	0.375
test	-3.359	-3.359	-3.359	0.278	1.0e-05	-0.043

The prediction results for node 0x0A0AC927 in the PGP network suggest that, while certain features retain limited predictive capacity, the overall time series is characterized by high instability and low predictability.

In terms of point-wise similarity, *popularity* achieves consistent R^2 values on both the training and test sets (0.423 and 0.409), indicating stable performance. In contrast, *kappa* performs well on the training set ($R^2 = 0.766$) but collapses on the test set with a highly negative R^2 of -170.82 and extreme bias (MBD = 154.77), reflecting total model failure. The *similarity* feature has a moderate training R^2 of 0.375, but it drops to -3.359 in the test set, again indicating poor generalization.

Regarding trend similarity, *popularity* maintains a relatively stable correlation on the test set (0.678), while *similarity* shows moderate trend alignment (0.278). *Kappa*, however, reverses its trend on the test set with a correlation of -0.286.

For shape similarity, the DTW values increase significantly in the test set across all features, especially for *kappa* (53.05), indicating severe shape misalignment. Sequence Consistency scores drop below zero for all features in the test set, with *similarity* falling from 0.375 to -0.043, suggesting unstable local trend directionality.

Overall, although *popularity* retains some predictive ability on the test set, both *kappa* and *similarity* degrade substantially. The time series of this node exhibits strong non-stationarity and structural volatility, posing significant challenges to accurate forecasting.

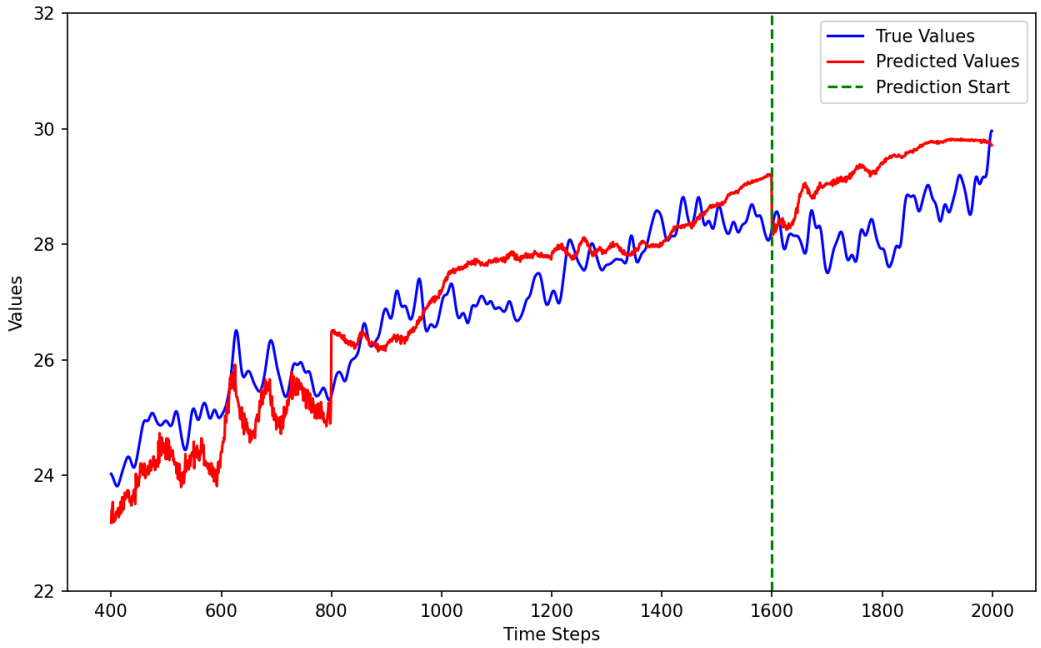

Figure 4.22: PGP, the node 0x0A2F87E5, The predicted(red) and real(blue) trajectories of popularity.

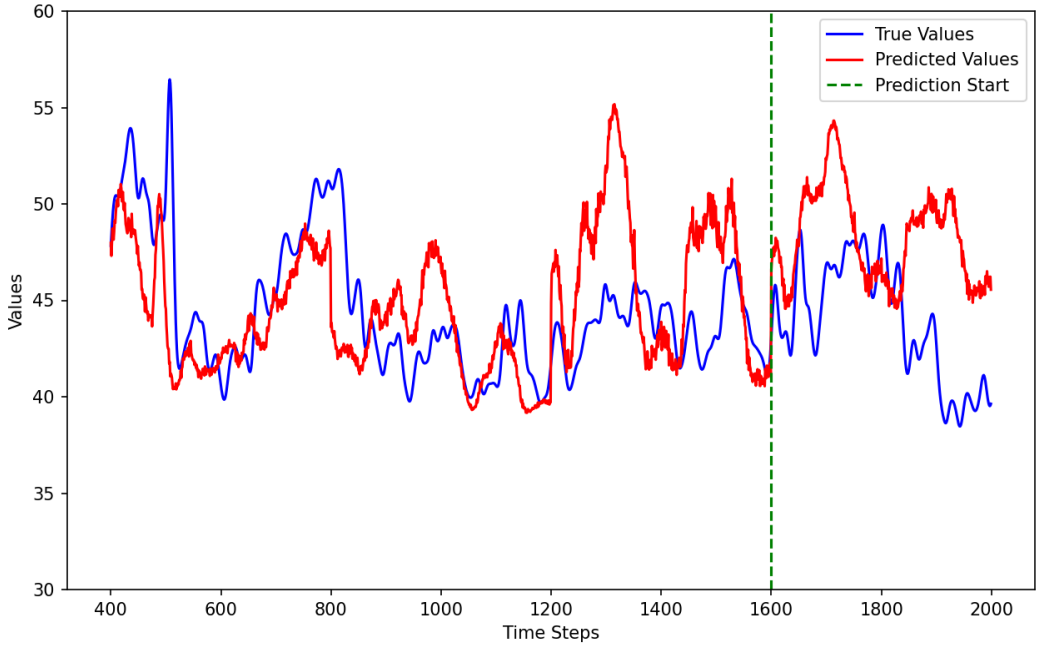

Figure 4.23: PGP, the node 0x0A2F87E5, The predicted(red) and real(blue) trajectories of expected degree .

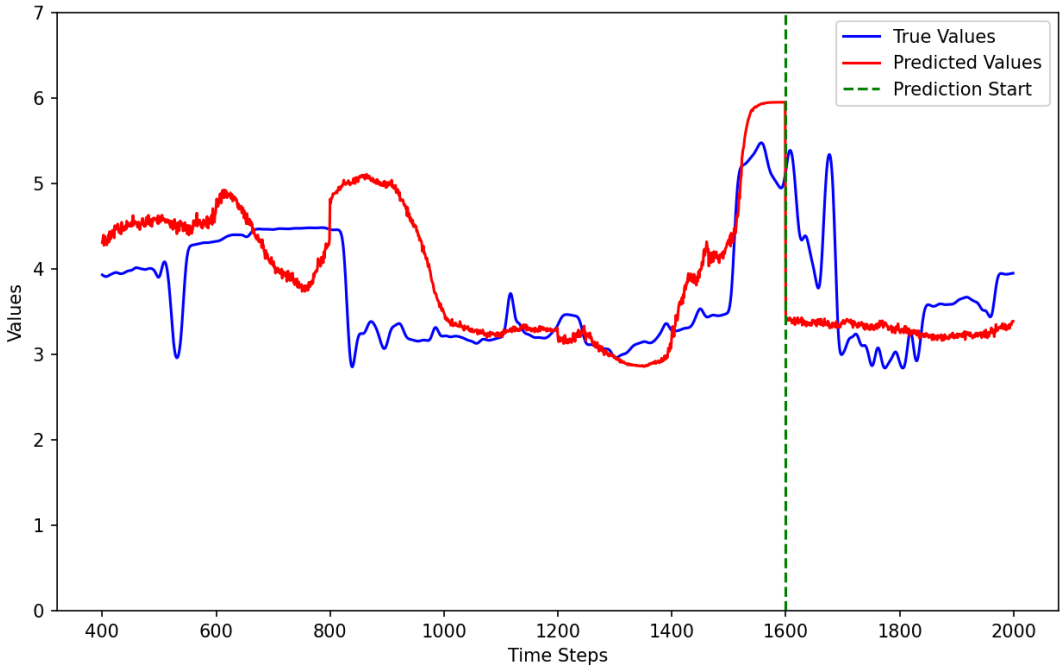

Figure 4.24: PGP, the node 0x0A2F87E5, The predicted(red) and real(blue) trajectories of similarity.

Table 4.8: PGP, the node 0x0A2F87E5

	R^2	MARE	MBD	Correlation	DTW	Sequence Consistency
popularity						
train	0.782	0.019	-0.016	0.948	2.3e-06	0.0408
test	-3.563	0.033	0.935	0.521	1.0e-05	-0.066
kappa						
train	-0.248	0.0623	0.220	0.391	1.4e-06	0.041
test	-2.435	0.113	4.402	0.159	5.1e-05	0.030
similarity						
train	-0.206	0.1453	0.350	0.658	3.7e-07	0.0381
test	-0.208	0.136	-0.333	0.321	2.3e-06	-0.006

The prediction results for node 0x0A2F87E5 in the PGP network demonstrate low overall predictability, particularly on the test set, where the model exhibits instability in trend consistency and structural alignment.

In terms of point-wise similarity, *popularity* performs well on the training set with an R^2 of 0.782, but this sharply declines to -3.563 on the test set, indicating a complete breakdown in generalization. Both *kappa* and *similarity* show negative R^2 values in both training and test sets, suggesting the model fails to learn meaningful numerical patterns for these features.

Regarding trend similarity, only *popularity* maintains a moderate test correlation (0.521), while *kappa* and *similarity* remain weak at 0.159 and 0.321, respectively, indicating that predicted sequences do not align well with the true trend directions.

For shape similarity, DTW values are within acceptable ranges, but Sequence Consistency drops significantly in the test set. *Popularity* and *similarity* show negative consistency values (-0.066 and -0.006), indicating disordered local trends.

Overall, while the model fits *popularity* well on training data, its performance on unseen data is poor across all features. Node 0x0A2F87E5's time series is marked by non-stationary behavior, weak trend alignment, and structural irregularity, making accurate forecasting difficult.

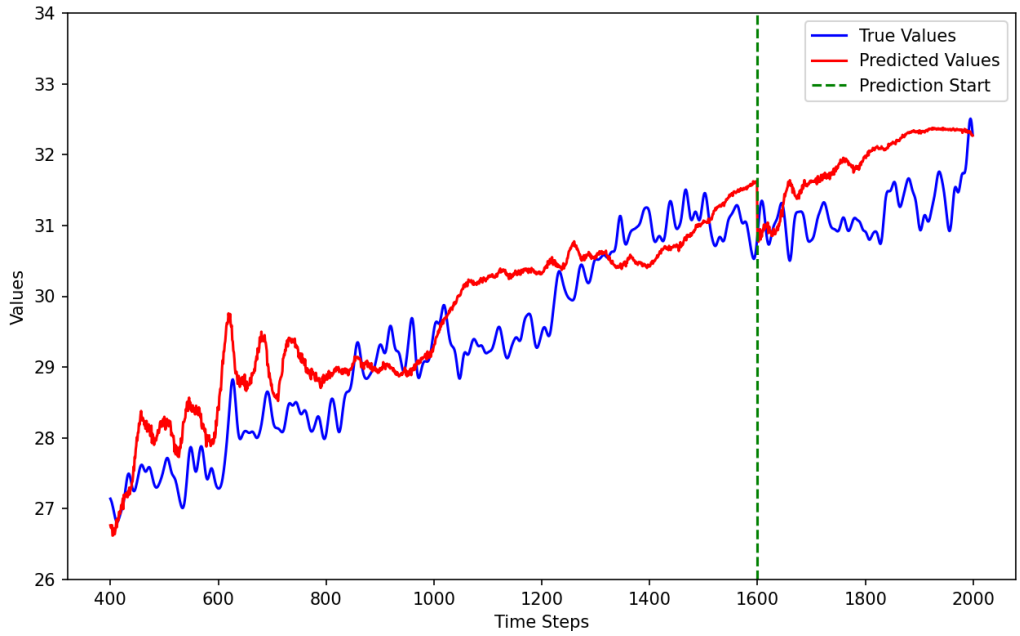

Figure 4.25: PGP, the node 0x0A15BE0D, The predicted(red) and real(blue) trajectories of popularity.

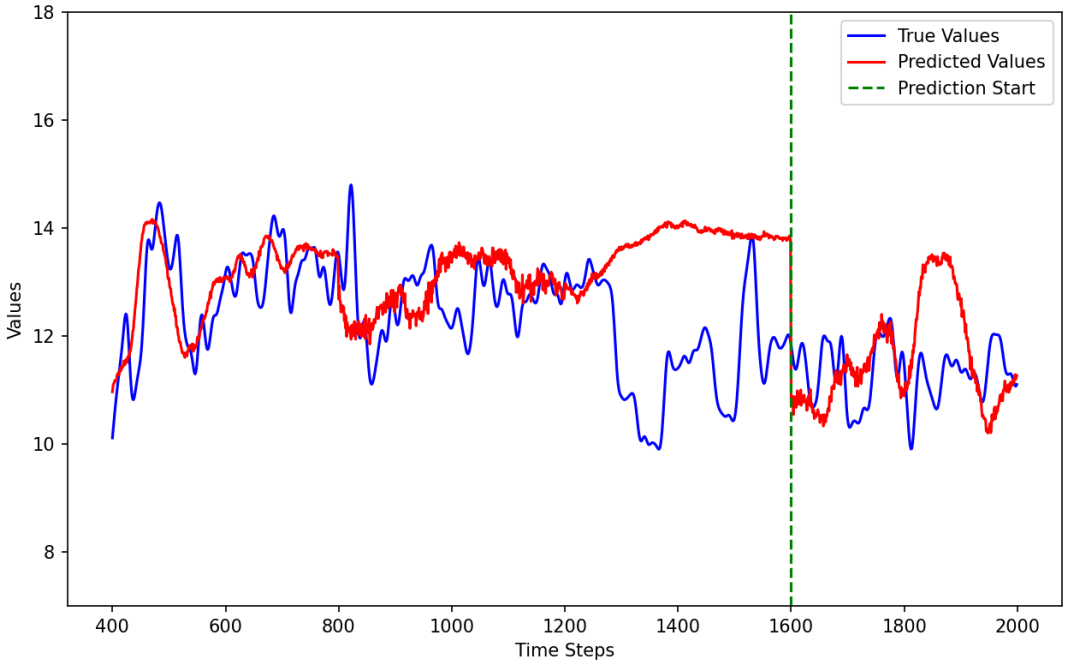

Figure 4.26: PGP, the node 0x0A15BE0D, The predicted(red) and real(blue) trajectories of expected degree similarity.

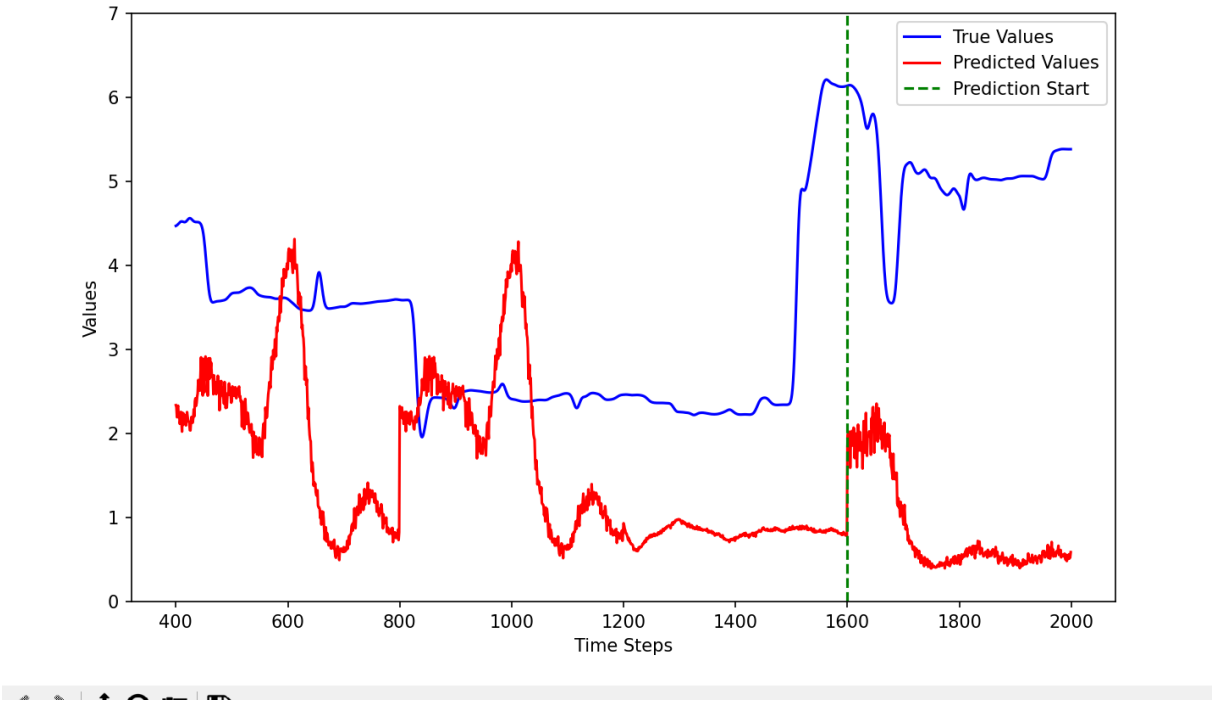

Figure 4.27: PGP, the node 0x0A15BE0D, The predicted(red) and real(blue) trajectories of similarity.

Table 4.9: PGP, the node 0x0A15BE0D

	R^2	MARE	MBD	Correlation	DTW	Sequence Consistency
popularity						
train	0.742	0.018	0.367	0.909	9.0e-07	0.064
test	-4.715	0.023	0.696	0.439	8.4e-06	-0.068
kappa						
train	-1.245	0.092	0.781	-0.152	1.1e-06	0.030
test	-3.598	0.079	0.280	-0.036	1.0e-05	0.049
similarity						
train	-1.820	0.327	-1.111	-0.264	1.0e-07	-0.002
test	-40.69	0.614	-3.144	-0.013	1.0e-05	0.038

The prediction results for node 0x0A15BE0D in the PGP network indicate extremely poor predictability, especially on the test set, where all features exhibit significant numerical errors, loss of trend direction, and structural instability.

In terms of point-wise similarity, *popularity* performs well on the training set ($R^2 = 0.742$), but its test performance deteriorates sharply ($R^2 = -4.715$), reflecting a complete breakdown in generalization. Both *kappa* and *similarity* show negative R^2 values across both training and test sets, with *similarity* reaching as low as -40.69 on the test set.

Regarding trend similarity, *popularity* maintains strong correlation on the training set (0.909), but this drops significantly to 0.439 in the test set. *Kappa* and *similarity* exhibit near-zero or negative correlation throughout, indicating that the model fails to capture trend directions and may even predict opposite

trends.

For shape similarity, although DTW values remain within acceptable ranges, Sequence Consistency scores are low on the test set. Notably, *popularity*'s consistency drops from 0.064 to -0.068, indicating disordered local trend alignment.

Overall, while the model fits *popularity* to some extent during training, its generalization across all features is extremely limited. The time series of node 0x0A15BE0D displays strong non-stationarity and structural volatility, making accurate prediction highly challenging.

4.1 Popularity

The popularity trajectories used in this study generally show an upward trend over time. However, in the short term, there are significant fluctuations with large variations. While the overall trend of the trajectories shows clear fluctuations, the long-term growth trend is more prominent. Among these, the popularity trajectories of the arXiv and PGP datasets have a more stable growth rate, while the popularity trajectory of the USAir dataset shows unstable growth with significant fluctuations of sharp increases and decreases over a relatively long period. We observe that the vast majority of nodes in real-world networks exhibit a long-term upward trend in popularity, which may be related to the growth in network size and the increased activity across the entire network.

In terms of prediction performance, the model performed well during training, closely following the overall trend of the real data, and it also captured some of the fluctuations in the details. However, during the testing phase, the performance clearly declined compared to the training phase. Given the clear upward trend in the data, the model made reasonable predictions regarding this trend. We can see that for the arXiv dataset, with the most stable growth rate, the predicted upward trend closely matched the real data. For the PGP and USAir datasets, although the model also predicted the upward trend, the speed of growth deviated to some extent from the real situation. The model, based on its performance during training, tried to predict these details and fluctuations using the learned patterns. However, from the prediction results, it seems the model has not yet accurately predicted these details, as the fluctuations and variations do not align well with the real data.

4.2 Expected degree

The expectation trajectories also exhibit significant short-term fluctuations, but from a long-term perspective, they lack a clear upward or downward trend, appearing more random compared to the popularity trajectories.

Similarly, during training, the model effectively learned the fluctuation characteristics and some details of the data, with the predicted results on the training set closely matching the real data. During testing, the model made similar predictions based on the oscillation patterns learned during training, and the range of predicted values was close to the actual range of values. However, it also failed to match the details. For the expectation trajectories, we predicted similar shapes, but were unable to capture the finer details.

4.3 Similarity

The similarity trajectories are the smoothest, showing no high-frequency fluctuations in the short term, but no clear trend was observed either. From the prediction results, the prediction performance for the similarity trajectories is the worst. The similarity trajectories performed poorly during training, and the predicted trajectory could not closely follow the real trajectory. The prediction results also deviated significantly from the actual values.

5 Conclusion and Recommendations

Through the analysis of prediction results, we observe that time series with a single clear trend—such as consistently increasing or decreasing trajectories—tend to exhibit the best prediction performance. For example, in the case of the popularity trajectory, where a long-term upward trend is evident, the model's predictions also generally follow an increasing pattern. Moreover, the model is capable of capturing patterns that resemble the actual data. However, regardless of whether the feature is popularity, expectation, or similarity, the model fails to make accurate predictions at a detailed level. In other words, although the model attempts to capture fine-grained variations, the predicted details are highly random and fall short of the performance achieved during training.

Interestingly, we find that the expectation trajectory of the ANC airport in the USAir dataset exhibits a noticeable periodic pattern. We conducted a separate prediction experiment on this specific time series, and the results show that although some amplitude deviations exist, the overall trend direction, turning points, and frequency closely match the actual data. This suggests that trends with apparent regular variation can also be effectively captured by the model, enabling reasonably accurate forecasts.

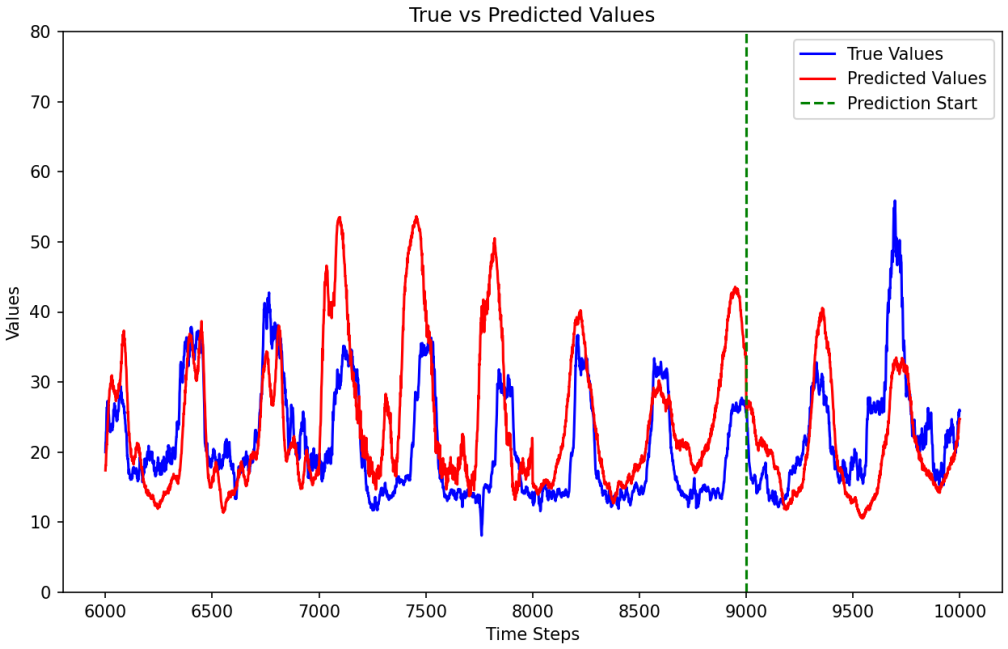

Figure 5.1: arXiv, the node ANC, The predicted(red) and real(blue) trajectories of expected degree.

In addition to the aforementioned sequences with strong regularity, the dataset also contains a large number of trajectories exhibiting continuously changing trends and frequent directional shifts. These time series often lack discernible patterns upon visual inspection—for instance, the similarity trajectories and the vast majority of expectation trajectories do not present any clear trend regularity.

Although neural networks are capable of fitting such fluctuating trends effectively on the training set and generating predictions with similarly dynamic behavior, this phenomenon raises two fundamental ques-

tions: (1) Does the strong performance on the training set reflect the model's ability to learn underlying structural patterns in the data, or is it merely a result of memorizing the training examples? (2) Is the poor generalization performance on the test set a consequence of overfitting, or does it stem from an inherent lack of predictability in the data itself?

At the core of these questions lies a deeper issue: whether such shifting trends are governed by any latent patterns, and whether neural networks possess the capacity to discover them. Importantly, these questions are not mutually exclusive nor strictly separable, as they often lead to the same observable outcome—limited predictive accuracy on unseen data. Therefore, further investigation of these non-stationary sequences may help delineate the boundaries and limitations of neural networks in time series forecasting tasks.

First, the hyperparameter configurations we adopted (such as embedding dimension, number of attention heads, and feedforward network width) are commonly used and widely validated in the literature, and are considered generally reasonable. Second, we confirmed that the model produces valid and reliable predictions on structured time series, such as the expectation trajectory of the ANC airport in the USAir network, which exhibits clear periodic patterns.

However, when switching to other datasets, we observed that despite the model performing well on the training set—with the loss decreasing steadily—the validation loss remained highly volatile and failed to converge. Notably, across multiple experiments, we observed significant random fluctuations: even within a reasonable range of hyperparameters, the training results varied substantially from run to run. This randomness masked the effects of fine-tuning and prevented the identification of a stable, optimal parameter combination. As a result, we were only able to determine a "reasonable parameter range," within which tuning efforts had little impact on validation performance, which appeared to be limited by the inherent predictability of the data itself.

Furthermore, upon visual inspection, we found that these time series lacked clear trend structures or identifiable patterns—making it difficult, even for human observers, to discern any consistent or meaningful progression. Although the model demonstrated strong fitting ability on the training data and poor performance on the validation data—an observation that superficially resembles overfitting—we found that neither modifying the model architecture nor adjusting the hyperparameter space improved the outcome. In the absence of discernible trends or periodic structures in the data, we conclude that the failure to generalize is not due to overfitting in the conventional sense, but rather stems from the intrinsic structural complexity and low predictability of the time series themselves.

We extracted the prediction results for nodes LAX, 1674, and 0x0D62001B from the USAir, arXiv, and PGP datasets in reference [1]. We found that traditional methods have the ability to fit details during the training phase, but the results in the prediction phase tend to become linear. Neural networks, in contrast, have stronger capabilities in both detail fitting and prediction, producing more complex results. Neural networks perform well in predicting the periodic components of the trajectory, but their accuracy in predicting details still needs improvement.

Compared to traditional methods, neural networks can automatically extract patterns from the data without relying on parameter estimation. Furthermore, neural networks have stronger learning capabilities for complex patterns in the data, allowing them to make predictions that include both trends and details.

However, neural networks cannot make predictions for extremely long time periods as traditional methods can. One issue is the lack of sufficient data, as neural networks often rely on large amounts of data. Additionally, they are more prone to problems like gradient vanishing and gradient explosion. Extremely long time series typically require deep neural networks, which result in a significant increase in parameters. Predicting sequences longer than 1000 steps is already a rare task in deep learning, and predicting even longer sequences would pose a significant challenge.

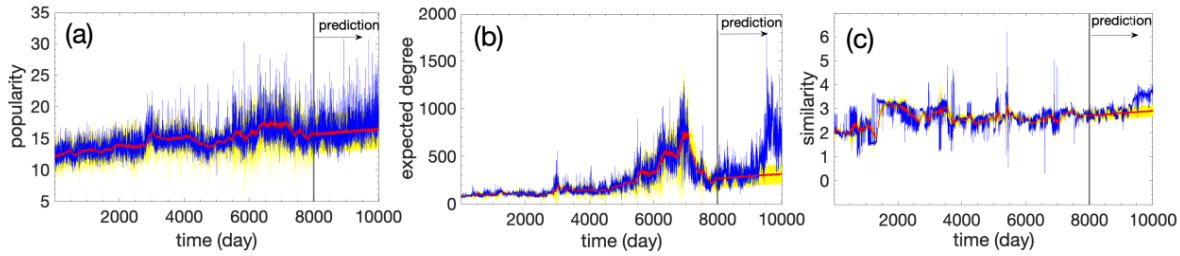

Figure 5.2: arXiv, the node LAX, The predicted(red) and real(blue) trajectories[1]

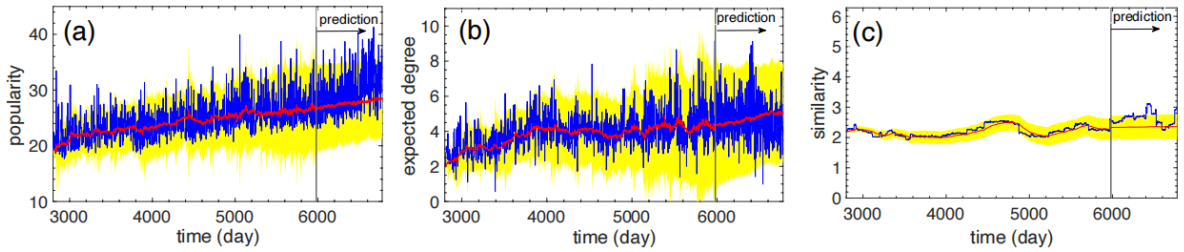

Figure 5.3: arXiv, the node 1674, The predicted(red) and real(blue) trajectories[1].

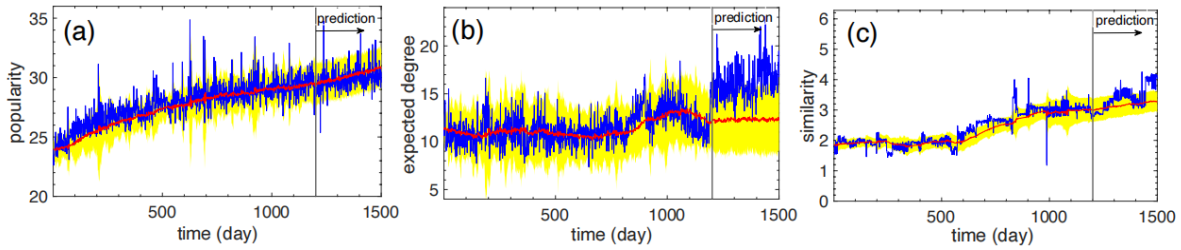

Figure 5.4: PGP , the node 0x0D62001B, The predicted(red) and real(blue) trajectories[1]

Based on the above prediction results, the neural network's ability to predict time series primarily depends on the periodic and trend components, while the noise components in the data are unpredictable. The current methods have successfully predicted the periodic and long-term increasing or decreasing trends. However, the performance of these methods is still limited in identifying more complex patterns in the sequence.

Currently, long-term accurate prediction tasks in deep learning face multiple challenges, including the long-term dependencies in the data, noise, and sudden value changes caused by unexpected events, all of which greatly interfere with prediction accuracy. Additionally, as the data increases, the computational

complexity of long-term sequence prediction significantly grows, making the need for large amounts of computational resources and time a major issue. In recent years, with the rapid development of deep learning, more and more models for time series modeling have emerged. These methods, by incorporating mechanisms such as sparse attention, seasonal modeling, and frequency domain modeling, have enabled deeper analysis of time series from various perspectives. In the future, these more complex models can be considered to better explore the potential complex patterns in these trajectories.

BIBLIOGRAPHY

[1] E. S. Papaefthymiou, C. Iordanou, and F. Papadopoulos, "Fundamental dynamics of popularity-similarity trajectories in real networks," *Physical review letters*, vol. 132, no. 25, p. 257401, 2024.

[2] F. Lorrain and H. C. White, "Structural equivalence of individuals in social networks," *The Journal of mathematical sociology*, vol. 1, no. 1, pp. 49–80, 1971.

[3] L. A. Adamic and E. Adar, "Friends and neighbors on the web," *Social networks*, vol. 25, no. 3, pp. 211–230, 2003.

[4] Q. Ou, Y.-D. Jin, T. Zhou, B.-H. Wang, and B.-Q. Yin, "Power-law strength-degree correlation from resource-allocation dynamics on weighted networks," *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics*, vol. 75, no. 2, p. 021102, 2007.

[5] T. Zhou, L. Lü, and Y.-C. Zhang, "Predicting missing links via local information," *The European Physical Journal B*, vol. 71, pp. 623–630, 2009.

[6] E. Ravasz, A. L. Somera, D. A. Mongru, Z. N. Oltvai, and A.-L. Barabási, "Hierarchical organization of modularity in metabolic networks," *science*, vol. 297, no. 5586, pp. 1551–1555, 2002.

[7] G. G. Chowdhury, *Introduction to modern information retrieval*. Facet publishing, 2010.

[8] E. A. Leicht, P. Holme, and M. E. Newman, "Vertex similarity in networks," *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics*, vol. 73, no. 2, p. 026120, 2006.

[9] V. Verma and R. K. Aggarwal, "A comparative analysis of similarity measures akin to the jaccard index in collaborative recommendations: empirical and theoretical perspective," *Social Network Analysis and Mining*, vol. 10, no. 1, p. 43, 2020.

[10] L. Lü, C.-H. Jin, and T. Zhou, "Similarity index based on local paths for link prediction of complex networks," *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics*, vol. 80, no. 4, p. 046122, 2009.

[11] M. Wallace, M. Shelkey *et al.*, "Katz index of independence in activities of daily living (adl)," *Urol Nurs*, vol. 27, no. 1, pp. 93–94, 2007.

[12] L. Pan, T. Zhou, L. Lü, and C.-K. Hu, "Predicting missing links and identifying spurious links via likelihood analysis," *Scientific reports*, vol. 6, no. 1, p. 22955, 2016.

[13] S. Gaucher and O. Klopp, "Maximum likelihood estimation of sparse networks with missing observations," *Journal of Statistical Planning and Inference*, vol. 215, pp. 299–329, 2021.

[14] L. Chai, L. Tu, X. Wang, and J. Chen, "Network-energy-based predictability and link-corrected prediction in complex networks," *Expert Systems with Applications*, vol. 207, p. 118005, 2022.

[15] E. J. Candès, X. Li, Y. Ma, and J. Wright, "Robust principal component analysis?" *Journal of the ACM (JACM)*, vol. 58, no. 3, pp. 1–37, 2011.

[16] G. Liu, Z. Lin, and Y. Yu, "Robust subspace segmentation by low-rank representation," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 663–670.

[17] L. Xing, B. Chen, J. Wang, S. Du, and J. Cao, "Robust high-order manifold constrained low rank representation for subspace clustering," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 2, pp. 533–545, 2020.

[18] R. Pech, D. Hao, Y.-L. Lee, Y. Yuan, and T. Zhou, "Link prediction via linear optimization," *Physica A: Statistical Mechanics and its Applications*, vol. 528, p. 121319, 2019.

[19] X. Xian, T. Wu, S. Qiao, X.-Z. Wang, W. Wang, and Y. Liu, "Netsre: Link predictability measuring and regulating," *Knowledge-based systems*, vol. 196, p. 105800, 2020.

[20] E. Nasiri, K. Berahmand, and Y. Li, "Robust graph regularization nonnegative matrix factorization for link prediction in attributed networks," *Multimedia Tools and Applications*, vol. 82, no. 3, pp. 3745–3768, 2023.

[21] N. M. Ahmed, L. Chen, Y. Wang, B. Li, Y. Li, and W. Liu, "Deepeye: Link prediction in dynamic networks based on non-negative matrix factorization," *Big Data Mining and Analytics*, vol. 1, no. 1, pp. 19–33, 2018.

[22] X. Shen and F.-L. Chung, "Deep network embedding for graph representation learning in signed networks," *IEEE transactions on cybernetics*, vol. 50, no. 4, pp. 1556–1568, 2018.

[23] J. Wang, J. Cao, W. Li, and S. Wang, "Cane: community-aware network embedding via adversarial training," *Knowledge and Information Systems*, vol. 63, pp. 411–438, 2021.

[24] R.-M. Cao, S.-Y. Liu, and X.-K. Xu, "Network embedding for link prediction: The pitfall and improvement," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 29, no. 10, 2019.

[25] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 701–710.

[26] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, 2013.

[27] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855–864.

[28] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th international conference on world wide web*, 2015, pp. 1067–1077.

[29] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 1225–1234.

[30] M. Nickel and D. Kiela, "Poincaré embeddings for learning hierarchical representations," *Advances in neural information processing systems*, vol. 30, 2017.

[31] ——, "Learning continuous hierarchies in the lorentz model of hyperbolic geometry," in *International conference on machine learning*. PMLR, 2018, pp. 3779–3788.

[32] F. Papadopoulos, M. Kitsak, M. □. Serrano, M. Boguñá, and D. Krioukov, "Popularity versus similarity in growing networks," vol. 489, no. 7417, pp. 537–540. [Online]. Available: https://doi.org/10.1038/nature11459

[33] F. Papadopoulos, C. Psomas, and D. Krioukov, "Network mapping by replaying hyperbolic growth," *IEEE/ACM Transactions on Networking*, vol. 23, no. 1, pp. 198–211, 2014.

[34] G. García-Pérez, A. Allard, M. Á. Serrano, and M. Boguñá, "Mercator: uncovering faithful hyperbolic embeddings of complex networks," *New Journal of Physics*, vol. 21, no. 12, p. 123033, 2019.

[35] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[36] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[37] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[38] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[39] Z. Fang, S. Tan, Y. Wang, and J. Lü, "Elementary subgraph features for link prediction with neural networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 4, pp. 3822–3831, 2021.

[40] L. Cai, J. Li, J. Wang, and S. Ji, "Line graph neural networks for link prediction," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 9, pp. 5103–5113, 2021.

[41] X. Liu, X. Li, G. Fiumara, and P. De Meo, "Link prediction approach combined graph neural network with capsule network," *Expert Systems with Applications*, vol. 212, p. 118737, 2023.

[42] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 04, 2020, pp. 3438–3445.

[43] N. Yadati, M. Nimishakavi, P. Yadav, V. Nitin, A. Louis, and P. Talukdar, "Hypergcn: A new method for training graph convolutional networks on hypergraphs," *Advances in neural information processing systems*, vol. 32, 2019.

[44] Y. Feng, H. You, Z. Zhang, R. Ji, and Y. Gao, "Hypergraph neural networks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 3558–3565.

[45] Y. Gao, Y. Feng, S. Ji, and R. Ji, "Hgnn+: General hypergraph neural networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 3, pp. 3181–3199, 2022.

[46] Y. Dong, W. Sawin, and Y. Bengio, "Hnhn: Hypergraph networks with hyperedge neurons," *arXiv preprint arXiv:2006.12278*, 2020.

[47] S. Bai, F. Zhang, and P. H. Torr, "Hypergraph convolution and hypergraph attention," *Pattern Recognition*, vol. 110, p. 107637, 2021.

[48] H. Fan, F. Zhang, Y. Wei, Z. Li, C. Zou, Y. Gao, and Q. Dai, "Heterogeneous hypergraph variational autoencoder for link prediction," *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 8, pp. 4125–4138, 2021.

[49] Q. Zhang, T. Tong, and S. Wu, "Hybrid link prediction via model averaging," *Physica A: Statistical Mechanics and its Applications*, vol. 556, p. 124772, 2020.

[50] S. S. Singh, D. Srivastva, A. Kumar, and V. Srivastava, "Flp-id: Fuzzy-based link prediction in multiplex social networks using information diffusion perspective," *Knowledge-Based Systems*, vol. 248, p. 108821, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950705122003859

[51] D. Lai, X. Shu, and C. Nardini, "Link prediction in complex networks via modularity-based belief propagation," *Chinese Physics B*, vol. 26, no. 3, p. 038902, 2017.

[52] E. Rossi, B. Chambers, M. Bronstein, D. Eynard, and F. Monti, "Temporal graph networks for deep learning on dynamic graphs," *arXiv preprint arXiv:2006.10637*, 2020.

[53] R. Trivedi, M. Farajtabar, P. Biswal, and H. Zha, "Dyrep: Learning representations over dynamic graphs," in *Proceedings of the 28th International World Wide Web Conference (WWW)*. ACM, 2019, pp. 1085–1095.

[54] A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, T. B. Schardl, and C. E. Leiserson, "Evolvegcn: Evolving graph convolutional networks for dynamic graphs," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 5363–5370.

[55] L. Yin, H. Zheng, T. Bian, and Y. Deng, "An evidential link prediction method and link predictability based on shannon entropy," *Physica A: Statistical Mechanics and its Applications*, vol. 482, pp. 699–712, 2017.

[56] X. Chen, L. Fang, T. Yang, J. Yang, Z. Bao, D. Wu, and J. Zhao, "The application of degree related clustering coefficient in estimating the link predictability and predicting missing links of networks," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 29, no. 5, 2019.

[57] X. Chen, P. Jiao, Y. Yu, X. Li, and M. Tang, "Toward link predictability of bipartite networks based on structural enhancement and structural perturbation," *Physica A: Statistical Mechanics and its Applications*, vol. 527, p. 121072, 2019.

[58] J. Sun, L. Feng, J. Xie, X. Ma, D. Wang, and Y. Hu, "Revealing the predictability of intrinsic structure in complex networks," *Nature communications*, vol. 11, no. 1, p. 574, 2020.

[59] T. Suo-Yi, Q. Ming-Ze, W. Jun, and L. Xin, "Link predictability of complex network from spectrum perspective," 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:219656149

[60] U.S. Department of Transportation, "On-time: Reporting carrier on-time performance (1987–present)," https://www.transtats.bts.gov/DL_SelectFields.aspx?gnoyr_VQ=FGJ&QO_fu146_anzr=b0-gvzr, 2023, accessed: July, 2023.

[61] Cornell University, "Kaggle - arxiv dataset," https://www.kaggle.com/datasets/Cornell-University/arxiv, 2023, accessed: July, 2023.

[62] Johan Cedergren, "Openpgp web of trust dataset," https://www.lysator.liu.se/~jc/wotsap/wots2/, 2023, accessed: July, 2023.

[63] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time Series Analysis: Forecasting and Control*, 5th ed. Wiley, 2015.

[64] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[65] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[66] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 785–794.

[67] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[68] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder–decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[69] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[70] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[71] S. Li, X. Jin, Y. Xuan, X. Zhou, W. Chen, Y. Wang, and X. Yan, "Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[72] B. Lim, S. O. Arik, N. Loeff, and T. Pfister, "Temporal fusion transformers for interpretable multi-horizon time series forecasting," *International Journal of Forecasting*, 2021.

[73] P. Lara-Benitez, L. Gallego-Ledesma, M. Carranza-García, and J. M. Luna-Romera, "Evaluation of the transformer architecture for univariate time series forecasting," *Applied Sciences*, vol. 11, no. 3, p. 808, 2021.

[74] E. Lezmi and J. Xu, "Time series forecasting with transformer models and application to asset management," *arXiv preprint arXiv:2302.10204*, 2023.

[75] Y. Liu, L. Liu, J. Zhang, T. Gao, H. Guo, X. Wang, Y. Zhang, J. Song, and L. Sun, "General time transformer for zero-shot multivariate forecasting," in *Proceedings of the ACM Web Conference 2024*. ACM, 2024, pp. 3734–3743.

[76] L. Sasal, C. Rupprecht, and S. Mandt, "W-transformers: Wavelet transformers for non-stationary time series forecasting," *arXiv preprint arXiv:2209.03945*, 2022.

[77] Y. Liu, J. Zhang, Y. Zhang, J. Song, L. Sun *et al.*, "itransformer: Inverted transformers are effective for time series forecasting," *arXiv preprint arXiv:2310.06625*, 2023.

[78] H. Wu, J. Xu, J. Wang, and M. Long, "Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting," *Advances in neural information processing systems*, vol. 34, pp. 22 419–22 430, 2021.

[79] Z. Dong, M. Yang, J. Wang, H. Wang, C. S. Lai, and X. Ji, "Pffn: A parallel feature fusion network for remaining useful life early prediction of lithium-ion battery," *IEEE Transactions on Transportation Electrification*, 2024.