UTS UNIVERSITY OF TECHNOLOGY SYDNEY

Autonomous Learning for Multiple Data Streams under Concept Drift

by Ming Zhou

Thesis submitted in fulfilment of the requirements for the degree of

Doctor of Philosophy in Computer Science

under the supervision of Distinguished Professor Jie Lu, A/Prof Guangquan Zhang and Dr Yiliao Song

University of Technology Sydney
Faculty of Engineering and Information Technology

June 2025

CERTIFICATE OF ORIGINAL AUTHORSHIP

I, *Ming Zhou*, declare that this thesis is submitted in fulfilment of the requirements for the award of *Doctor of Philosophy*, in the *School of Computer Science*, *Faculty of Engineering and Information Technology* at the University of Technology Sydney. This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis. This document has not been submitted for qualifications at any other academic institution. This research is supported by the Australian Government Research Training Program.

SIGNATURE: Production Note: Signature removed prior to publication.

DATE: 11th June, 2025

PLACE: Sydney, Australia

Abstract

Concept drift has always been a major challenge in data stream mining research, because the difference in data distribution caused by drift will affect the prediction performance. The adaptive learning of concept drift can help solve this problem. Currently, many adaptive learning frameworks for concept drift have been developed to make efficient and accurate predictions on data streams, but most of these studies focus on single data stream and rarely consider multiple data streams. In the multi-stream environment, the drift situation is more complicated, and the correlation between data streams needs to be taken into account. And the relationship between drifts also need to be further discussed.

In this research, we focus on the concept drift problem on multiple data streams and consider a novel, practical problem, termed *Autonomous Learning for Multiple Data Streams under Concept Drift* (ALMCD). To solve ALMCD problem, this thesis aims to develop a set of algorithms targeted to five orthogonal problems to improve the real-time prediction performance on multi-stream: 1) How to identify the concept drift problem on multiple data streams; 2) How to recognize the drift relationship between multiple data streams according to the concept drift in each data stream; 3) How to solve concurrent drift in multiple data streams with considering the drift relationship between streams; 4) How to solve delayed drift in multiple data stream with considering the relationship between streams; and 5) How to design autonomous learning algorithms for multiple data streams under concurrent/delayed drift.

To address Problem 1 and 2), this thesis develops the theoretical foundation for concept drift in multi-stream environments and provides detailed definitions (Chapter 3). These definitions ensure a focused analysis of adaptive learning for multi-stream concept drift, starting from concurrent drift scenarios. In Chapter 4, we refine the definition of inter-stream correlations in multi-stream environments and further identify their role in constructing multi-stream adaptive frameworks.

To address Problem 3), building on the relevant definitions, we develop a novel Bayesian network-based multi-stream adaptive framework. This framework tackles the challenge of concurrent drift in multi-stream environments by learning and transferring drift information from base streams (Chapter 3).

To address Problem 4, this thesis presents a novel adaptive framework designed for large-scale multi-stream environments. Both frameworks are constructed based on graph neural networks, where the graph structure is designed to represent inter-stream correlations. The first framework employs multi-model rolling online learning to maintain adaptability to new data distributions (Chapter 4).

To preliminarily address Problem 5, this thesis proposes a novel concept drift self-adaptation framework for multi-stream based on dynamic graph regularization. It enhances adaptability by considering the dynamic adjustment of the graph structure and imposing graph-based regularization constraints on the online autonomous adaptation learning process (Chapter 5).

To further address Problem 5, we consider more challenging multi-stream scenarios and the autonomous learning capabilities of models. In Chapter 6, we develop a multi-stream adaptive framework based on continuous graph learning, driven by a small amount of historical data. The adaptive graph generator operates during both model initialization and online adaptation phases, while the adaptive diffusion attention module captures changes in inter-stream correlations. Chapter 7 introduces a multi-scale

adaptive convolutional graph, leveraging convolutional structures at multiple scales to construct highly generalizable correlation graphs. A drift-matching module is incorporated to proactively predict potential recurring drift patterns and prevent catastrophic forgetting.

In summary, this thesis addresses the problem of multi-stream concept drift by leveraging the identification of inter-stream correlations. It proposes a set of effective algorithms to enhance the generalization and learnability of the initialized model while fully utilizing the advantages of correlation structures to facilitate autonomous online adaptive learning.

Dedication

To myself . . .

ACKNOWLEDGMENTS

I would like to extend my heartfelt gratitude to everyone who supported me during my Ph.D. journey at the University of Technology Sydney (UTS). Although this journey was inherently challenging, the unwavering support of those around me made these four years not only manageable but also an unforgettable experience.

First and foremost, I would like to express my sincere gratitude to my supervisors: my principal supervisor, Distinguished Professor Jie Lu, and my co-supervisors, Associate Professor Guangquan Zhang and Dr. Yiliao Song. Their enthusiasm, patience, and insightful guidance were instrumental in shaping my research. In particular, Professor Jie Lu played a pivotal role in introducing me to the world of academia and establishing a rigorous and systematic approach to scientific research. Her encouragement and confidence during challenging times, coupled with her continuous support for my academic and personal growth, were invaluable. If my Ph.D. journey were akin to navigating the vast ocean, she served as the compass, providing direction and far-sighted advice that guided me to meaningful discoveries.

I am also profoundly grateful to the University of Technology Sydney and the Australian Research Council (grant under FL190100149) for providing the financial support necessary for my studies and life in Sydney.

My heartfelt thanks go to all members of the Decision Systems & e-Service Intelligence Lab (DeSI) within the Australian Artificial Intelligence Institute (AAII). Together, we cultivated a collaborative, vibrant, and motivating research environment, and it

has been my privilege to share this journey with you all. I am particularly grateful to Guangzhi Ma, Dr. Keqiuyin Li, Pengqian Lu, and Dr. Yi Zhang for their constructive advice on my research. My thanks also go to Yue Yang, Dr. Mengjia, Wei Duan, Xinheng Wu, Dr. Bin Zhang, Zihe Liu, and Dr. Kun Wang for their invaluable support in both my academic and personal endeavors.

I am deeply grateful to my parents. Their encouragement and confidence in me even before I embarked on this Ph.D. journey laid a solid foundation for my academic pursuits. Throughout these four years, their consistent support and unwavering belief in my passion for research kept me motivated to strive for excellence. As my earliest educators, they instilled in me a positive outlook on life and guided me with strong moral and ethical values. They have been my steadfast pillars, accompanying me in my growth and serving as my strongest support system.

I also wish to express my gratitude to my friends for their companionship throughout this journey. Together, we shared moments of joy and challenge, growing alongside one another, making this journey far less lonely.

Lastly, I extend my heartfelt thanks to my family and friends in China. Despite the physical distance, your care and unwavering support have been a constant source of strength and courage.

List of Publications

Published Papers:

1. **Ming Zhou**, Jie Lu, Continuous Graph Learning-based Self-adaptation for Multi-stream Concept Drift, *IEEE Transactions on Cybernetics* (IEEE-TCYB), DOI: 10.1109/TCYB.2025.3569816. [ERA&CORE: A, JCR Q1]

2. **Ming Zhou**, Jie Lu, Pengqian Lu, Guangquan Zhang, Dynamic Graph Regularization for Multi-Stream Concept Drift Self-Adaptation, *IEEE Transactions on Knowledge and Data Engineering* (IEEE-TKDE), Vol. 36, no. 11, pp. 6016-6028, Nov. 2024. DOI: 10.1109/TKDE.2024.3401156. [ERA&CORE: **A***, JCR Q1]

3. **Ming Zhou**, Jie Lu, Yiliao Song, Guangquan Zhang, Multi-Stream Concept Drift Self-Adaptation Using Graph Neural Network, *IEEE Transactions on Knowledge and Data Engineering* (IEEE-TKDE), Vol. 35, no. 12, pp. 12828-12841, 1 Dec. 2023. DOI: 10.1109/TKDE.2023.3272911. [ERA&CORE: **A***, JCR Q1]

4. **Ming Zhou**, Yiliao Song, Guangquan Zhang, Bin Zhang and Jie Lu, An Efficient Bayesian Neural Network for Multiple Data Streams, *Proceedings of the 2021 International Joint Conference on Neural Networks* (IJCNN 2021), pp. 1-8, 2021. [ERA&CORE: A]

Submitted Papers:

1. **Ming Zhou**, Jie Lu, Multi-Scale Adaptive Convolutional Graph for Multi-Stream Concept Drift, *IEEE Transactions on Cybernetics* (IEEE-TCYB). [ERA&CORE: A, JCR Q1] (Under Review)

Table of Contents

LIST OF FIGURES

FIGURE **Page**

List of Tables

Table **Page**

CHAPTER 1

INTRODUCTION

1.1 Background

The widespread use of mobile devices and the rapid expansion of information networks have led to an unprecedented increase in the generation of data [1]. With the ubiquitous presence of information perception and sensor networks, the speed at which data is generated has accelerated [2]. This information is often represented as streams, which are continuously updated [3, 4]. As a result, streaming data mining has become one of the most prominent areas of research in recent years [5]. The challenge of overcoming the inherent uncertainties in these streams and obtaining accurate real-time predictions has garnered increasing attention [6].

Traditional machine learning algorithms assume that the data stream is stationary, meaning that key statistical properties remain relatively constant over time. For stationary data streams, the training set and the test set are assumed to follow the same probability distribution [7]. Therefore, the model that learns the probability distribution of the historical data can forecast the target variables for future data [8]. However, this

assumption of stationarity is limited in real-world applications, where data streams are often non-stationary and subject to concept drift [9].

Concept drift describes the phenomenon of changes in statistical variables or data distribution in an unstable data stream, and this phenomenon will occur unpredictably over time [10]. When the relevance of the newly generated data and the old data is greatly weakened, some inductive methods in a stationary data environment will not be able to adapt, resulting in poor or even completely deviating forecast results from the actual trend [11]. In diverse fields such as social networks, financial systems, transportation, and medical monitoring, concept drift is a frequent occurrence, driven by rapidly changing topics, evolving products, fluctuating markets, and the dynamic effects of various unforeseen events.

Concept drift refers to the unpredictable changes in statistical variables or data distribution within an unstable data stream over time [10]. Such drifts weaken the relevance between newly generated data and historical data, rendering traditional inductive methods designed for stationary environments ineffective. This often leads to inaccurate or entirely misaligned predictions compared to actual trends [11]. Concept drift is particularly common in fields like social networks, financial systems, transportation, and medical monitoring, where rapidly evolving topics, shifting markets, product changes, and unexpected events drive continuous data variability.

Concept drift poses enduring challenges in detection, quantification, and adaptation within machine learning [12]. Drift detection focuses on identifying when drift occurs, often by monitoring learner error [13] or analyzing specific statistical measures [14]. Quantification involves determining the time, severity, and affected area of the drift. While most detection algorithms can estimate drift timing and severity, pinpointing the affected area often depends on the nature of the predictor [15]. Drift adaptation aims to integrate the statistical information from new data into the predictor to maintain or

enhance prediction accuracy. This is typically achieved through two strategies: retraining the model on new data to adjust to the changing distribution or incrementally updating the existing predictor using the new data [16, 17].

Over the past decade, concept drift learning has gained increasing attention, leading to the development of numerous detection and adaptive algorithms tailored to diverse scenarios. However, despite the growing interest, comprehensive studies on drift-handling techniques remain limited. This scarcity is partly due to the wide-ranging nature of concept drift, which spans multiple research domains.

1.2 Motivation

So far, the concept drift handling techniques have been validated in many single data stream cases [4, 18]. However, handling multiple data streams with concept drift is still an unsolved question. In addition, data streams in real scenarios basically do not exist independently, and data mining on multiple related data streams is the real dilemma [19]. Taking a traffic scene as an example, the bus routes of different lines in a city may overlap or intersect. The operating information of these lines is concurrent in time, and there is a certain degree of dependency between each other. Although many existing methods applicable to single streams can be applied repeatedly to multiple streams, this approach is rough and inefficient. The correlations between streams will be ignored, and isolated modelling cannot guarantee the synchronicity of tasks across multiple streams. Little research has been dedicated to addressing this gap. Consequently, in this paper, we focus on a more realistic and challenging problem named *Autonomous Learning for Multiple Data Streams under Concept Drift* (ALMCD). We consider a concept drift learning scenario across multiple streams, where the complex inter-stream correlation is taken into account and utilized as part of the model's driving mechanism to aid prediction and adaptation. In this process, the dynamic structural changes in inter-

stream correlation will be learnable, enabling the model to automatically capture data distribution changes caused by drift.

To solve the ALMCD problem and achieve better learning performance for multi-stream in practical scenarios, there are five orthogonal problems that need to be solved: 1) how to identify the concept drift problem on multiple data streams; 2) how to identify the concept drift problem on multiple data streams; 3) how to recognize the drift relationship between multiple data streams according to the concept drift in each data stream; 4) how to solve delayed drift in multiple data stream with considering the relationship between streams; 5) how to design autonomous learning algorithms for multiple data streams under concurrent/delayed drift. This thesis presents a thorough analysis of the aforementioned challenges and proposes solutions to address them.

1.3 Research Questions and Objectives

To handle the mentioned problems of autonomous learning for multiple data streams under concept drift and fill the research gaps, this research has the following research questions (**RQ**):

1.3.1 Research Questions

This research has five main Research Questions (RQ) as follows:

RQ1: How to identify the concept drift problem on multiple data streams?

When concept drift is discussed in the context of a single data stream, it typically refers to changes in the data distribution over time. However, when multiple data streams are involved, the problem becomes more complex. It is necessary to examine whether there are inter-stream drift dependencies—that is, whether the drift in one stream is influenced by the drift in another. If such dependencies exist, further analysis is required to understand various aspects of the drift, including the order of occurrence,

the magnitude of change, and the presence of common drift patterns across streams. Accurately identifying the types of drift is essential for developing effective adaptive learning algorithms.

RQ2: How to recognize the drift relationship between multiple data streams according to the concept drift in each data stream?

When discussing concept drift across multiple data streams, we focus on the scenario where there are drift relationships among the streams. In this context, there are typically two types of drift: concurrent drift and delayed drift. Concurrent drift occurs when the drifts in multiple data streams happen simultaneously due to the influence of a common factor, resulting in shared patterns of change across the streams. Delayed drift, on the other hand, refers to situations where the drift in one stream occurs after the drift in another. In this case, knowledge from streams where drift occurs earlier can be leveraged. Recognizing the drift relationships between multiple data streams is essential for tailoring the learning algorithm to the specific drift patterns.

RQ3: How to solve concurrent drift in multiple data streams with considering the drift relationship between streams?

Concurrent drift refers to the simultaneous occurrence of drift across multiple related data streams, with these drifts exhibiting strong similarities. While it is true that some prediction tasks can be addressed by applying single-stream adaptive methods to multiple streams, when rapid response and simultaneous predictions for multiple streams are required, methods capable of handling drift across streams need to be designed. These algorithms must consider how to establish connections between data streams that can be differentially mapped. Exploring how to effectively utilize these connections and integrate adaptive learning of concept drift across multiple data streams is an intriguing area of research.

RQ4: How to solve delayed drift in multiple data streams with considering the

relationship between streams?

The key difference between delayed drift and concurrent drift is that, in the case of delayed drift, there is a time gap between the occurrence of drift across multiple related data streams. Since the drifts do not occur simultaneously, we cannot synchronously map the drift at a particular moment when predicting across multiple streams. However, this time gap provides an advantage, as it allows us to leverage drift information from streams that have already experienced drift to predict those that have not yet drifted. Therefore, we must design algorithms that can utilize knowledge from streams where drift has occurred earlier. This also implies that our algorithms need to explore the details of the relationships between the drifts.

RQ5: How to design autonomous learning algorithms for multiple data streams under concurrent/delayed drift?

Concurrent drift and delayed drift, influenced by various factors, are the main types of drift observed across multiple related data streams. However, adaptive algorithms often struggle to meet the prediction demands of practical applications. The presence of non-ideal drift environments presents an even greater challenge. Drift types evolve over time, and additional uncertainties, such as noise and frequency differences between data streams, must also be accounted for. Therefore, we need to design a set of autonomous learning algorithms capable of automatically adapting to different types of drift (concurrent and delayed) across multiple data streams.

1.3.2 Research Objectives

To answer these research questions, we set up the corresponding Research Objectives (**RO**):

RO1: Develop a set of algorithms to identify the concept drift problem on multiple data streams and recognize the drift relationship between multiple data streams according to

the concept drift in each data stream (to answer RQ 1 and RQ 2).

When we do research on the concept drift of single data stream, we only need to focus on the nature of the current data stream itself. But for the concept drift on multiple data streams, we need to discuss whether there is a dependency between the data streams. If the data streams are not independent of each other, then we consider the relationship between drifts on multiple streams. We can further divide drift into two main types: concurrent drift and delayed drift. Therefore, it is necessary to develop a set of algorithms to mine the relationship between drifts to help us build efficient multi-stream adaptive prediction frameworks.

RO2: Develop a set of algorithms to solve concurrent drift in multiple data streams with considering the drift relationship between streams (to answer RQ 3).

Concurrent drift emphasizes that drift occurs on multiple data streams at the same time. In this case, the drifts is caused by a factor, and they would have certain commonalities. Simply copying the adaptive method for the single data stream to multiple data streams is difficult to meet the prediction requirements of practical applications, and will cause a large number of repeated calculations. To solve this problem, we need to develop a set of algorithms by mining the relationship between drifts and building connections between data streams. This enables the algorithms to efficiently learn and map when drift occurs, avoiding repeated model training and giving online predictions for multiple streams simultaneously.

RO3: Develop a set of algorithms to solve delayed drift in multiple data streams with considering the relationship between streams (to answer RQ 4).

When there is a time difference between the drifts that occurs on multiple data streams, we call it delayed drift. This means that the drifts of part of the data stream would happen earlier. In this case, we cannot map the drift content to all streams simultaneously when drift occurs. To solve this problem, we need to design a set of

algorithms to make full use of the knowledge from streams where their drift occurs earlier, so as to help other data streams to make predictions. At this time, the algorithms need to further explore the details of the relationship between drifts.

RO4: Develop a set of autonomous learning algorithms for multiple data streams under concurrent/delayed drift (to answer RQ 5).

Real scenes are more complex and changeable. Often the scenarios we encounter are not limited to a single drift type, and the previous drift type may also change over time. Therefore, the single adaptive algorithm can only handle an idealized multi-stream environment. In addition, there are some uncertain factors that may interfere with the prediction task, such as noise. To solve this problem, we need to develop a set of autonomous learning algorithms for multiple data streams under concurrent/delayed drift. The algorithms can automatically switch and adjust to accommodate the different types of drift on multiple streams.

1.4 Research Contribution

The main contributions of this thesis are summarised as follows:

Contribution 1. A novel adaptation framework using Bayesian neural networks is proposed for the concurrent drift problem in multiple data streams.

1) This study identifies the concept drift in multi-stream and discusses the relationships between drift based on different drift types.

2) This study is the first to build a concurrent drift adaptation framework for multiple data streams.

3) This study embeds Bayesian-based connectors to adapt to concurrent drift, named *BNN-connector*. It takes less time to update the connector than updating the predictor, the computational cost has been largely decreased when predicting multiple data streams simultaneously.

4) This study uses a single-stream adaptation method to update predictors for the base stream. Connectors can continuously learn the drift information from the base stream and use the learned information to correct the prediction results for other streams.

Contribution 2. A novel self-adaptation learning framework for multi-stream using graph neural network, termed SAGN is proposed to handle different types of drift in multi-stream.

1) This study develops a theoretical foundation for the correlation in multi-stream and its changes. The correlation contains the dependencies between streams and the correlation between drifts. Compared to previous research, it does not need to assume the source/target stream.

2) This study designs an online adaptation strategy for the GNN to handle different types of concept drift which is implemented locally in sub-graphs. An advantage of this design is that globally it maintains the correlation in multi-stream. The proposed dynamic validation set and GNN pool rolling mechanism are to work in tandem to achieve this process.

3) The proposed SAGN can overcome the concept drift problem in multi-stream while preserving the correlation between streams.

Contribution 3. A novel concept drift self-adaptation framework for multi-stream based on dynamic graph regularization, named MSGR, is developed to solve the concept drift adaptation learning problem further.

1) The graph-based regularization assigns dynamic learning weight for sub-graph updating according to the drift detection result, which can help the model adapt to the concept drift in multi-stream more effectively and targeted.

2) This study learns a correlation graph structure by our designed novel graph convolutional layer to capture the deep dependencies between streams. This graph structure is realized by embedding a sampling graph from a multi-stream historical data

distribution and calculating an adaptive transition matrix based on each node pair.

Contribution 4. A novel continuous graph learning-based self-adaptation framework for multi-stream concept drift, called CGLM. Designed for lifelong learning environments, CGLM demonstrates robust adaptive performance even when only limited historical data is available for model initialization.

1) This study designs a novel graph convolutional layer embedded in an Adaptive Graph Generator (AGG). This approach captures deep spatio-temporal relationships between streams without pre-defined graphs. We employ distinct continuous graph learning mechanisms tailored for both non-drift and drift scenarios, ensuring high-performance dynamic self-adaptation to concept drift.

2) The Adaptive Diffusion Graph Attention Module (AGDAT) is designed to capture local correlation changes for drift self-adaptation. Initially, AGG constructs a dynamic feature graph based on the newly arriving samples. Subsequently, AGDAT applies a multi-head diffusion attention mechanism to this feature graph, enabling adaptive sub-graph weight updating for the original correlation graph while maintaining global correlation stability.

3) when only small-scale data is available for model initialization, our proposed CGLM can achieve significant improvement over state-of-the-art baseline methods. This also demonstrates that CGLM maintains stable performance over long-term updates. Additionally, our method maintains leading prediction performance as the training data volume increases.

Contribution 5. A novel multi-scale adaptive convolutional graph framework, named MACG, is designed to improve the generalizability of the initialized model and enable proactive model updates to predict and adapt to drift during online testing.

1) This study performs multi-sampling on historical data across different time-scale

convolutions to initialize a highly generalizable correlation graph structure without pre-defined graphs during training. This approach captures various levels of spatial-temporal dependencies between streams, ensuring that the generated graph structure provides a more flexible and comprehensive representation of data characteristics.

2) The novel adaptation learning module in this study, is designed to facilitate model updates for effective drift adaptation. When drift is detected, MACG first generates a new correlation graph by applying the same multi-scale adaptive convolutions used during training to the feature sliding window, then fuses it with the previous graph. Next, MACG leverages historical data to match potential drift patterns and combines them with new samples and the fused graph to predict the current drift and update sub-graph weights. Furthermore, a shadow model synchronously updates using only new samples, while dynamic random validation mitigates the risk of erroneous updates caused by misinterpreted drift trends.

1.5 Research Significance

The significance and innovation of this survey will be summarized in this section. It will be described from two aspects: the theory significance and practical significance.

Theoretical significance: This thesis focuses on developing a set of autonomous learning algorithms for multiple data streams, extending traditional adaptive learning frameworks designed for single-stream scenarios. The goal is to address the challenges of concept drift in multi-stream environments. By analyzing various types of drift across multiple data streams, the relationships between these drifts are further examined. Adaptive algorithms are designed to cater to the unique characteristics of each drift type, enabling real-time prediction across multiple streams.

Firstly, a Bayesian neural network model is proposed to handle the concurrent drift problem in multi-stream, which leverages the inter-stream correlations to build

continuously updated connectors between streams. This integration allows the model to respond rapidly to concurrent drift during the autonomous learning process. Then, a set of GNN-based autonomous concept drift learning algorithms is developed. These algorithms utilize historical data to uncover correlations between streams and generate a base prediction model. The model represents the correlation between streams by a dynamic and learnable graph structure. When online testing starts, the model will learn the new data distribution information from newly arriving multi-stream samples and update the sub-graphs in the correlation structure to lead the model adaptation.

This design of correlation-based adaptation learning can make full use of relationships between streams and capture changes in multi-stream caused by concept drift. This design approach facilitates achieving a unified prediction frequency across multiple streams, which is particularly crucial when drift occurs. By treating the multiple streams as a whole, the autonomous adaptation process is transformed into sub-graph updates. This allows for handling localized drifts within the streams without compromising global stability.

Practical significance: The findings of this thesis contribute to practical applications in three key areas. First, we examine the various types of drift across multiple data streams, focusing on the dependencies between streams. Based on these relationships, we classify drifts in interconnected data streams into two main types: concurrent drift and delayed drift. Second, we design specific adaptive algorithms tailored to each type of drift by analyzing and learning the correlations between them. Third, we integrate these adaptive algorithms into the autonomous learning framework, addressing the diverse forecasting needs of complex real-world applications. Additionally, we extend the framework to account for other uncertainties in multiple data streams, offering valuable insights for adapting the proposed algorithms to more intricate scenarios.

1.6 Thesis Structure

The structure of the thesis is shown in Figure 1.1 and the chapters are organised as follows:

- CHAPTER 2 presents the literature review for concept drift and the graph neural networks modelling for multi-stream. Meanwhile, discusses limitations in the current research field.

- CHAPTER 3 presents 1) the definitions of the drift types in multi-stream and recognizes the drift relationship between multiple data streams; 2) a novel adaptation framework to handle the concurrent drift problem in multi-stream, which just embeds an adaptation method on one base stream and updates other streams by Bayesian-based connectors to reduce computational cost. This chapter addresses RQ1, RQ2 and RQ3 to achieve RO1 and RO2.

- CHAPTER 4 presents 1) the definitions of correlation and correlation changes in multi-stream and uses graph structure to present the correlation; 2) a novel self-adaptation framework for multi-stream based on graph neural networks. The concept drift problem is solved by rolling multiple GNNs and the designed online learning strategy. This chapter addresses RQ4 to achieve RO3.

- CHAPTER 5 presents a novel concept drift self-adaptation framework for multi-stream based on dynamic graph regularization. It can cover any drift type and is easy to apply to large-scale datasets. The adaptation process we design links the sub-graph updating with the degree of concept drift. The proposed graph-based regularization to sub-graph updating can help to capture the new correlation change caused by drifts. This chapter preliminarily addresses RQ5 to achieve RO4.

- CHAPTER 6 presents a novel continuous graph learning-based self-adaptation framework for multi-stream concept drift. It accounts for stricter multi-stream model initialization conditions and considers the framework's adaptability to concept drift from a lifelong learning perspective. It can initialize the model using only a small amount of historical data and maintain stable performance during long-term autonomous learning. This chapter further addresses RQ5 to achieve RO4.

- CHAPTER 7 presents a novel multi-scale adaptive convolutional graph framework. It approaches multi-step prediction by utilizing multi-scale time windows to capture the characteristics of multi-stream data. It enhances the initial model's generalization while balancing the prediction accuracy for both short-term and long-term horizons. During the online adaptation phase, it employs proactive prediction to match previously encountered drift patterns, improving adaptability while mitigating the risk of catastrophic forgetting. This chapter further addresses RQ5 to achieve RO4.

- CHAPTER 8 summarizes the key findings of this thesis and outlines potential directions for future research.

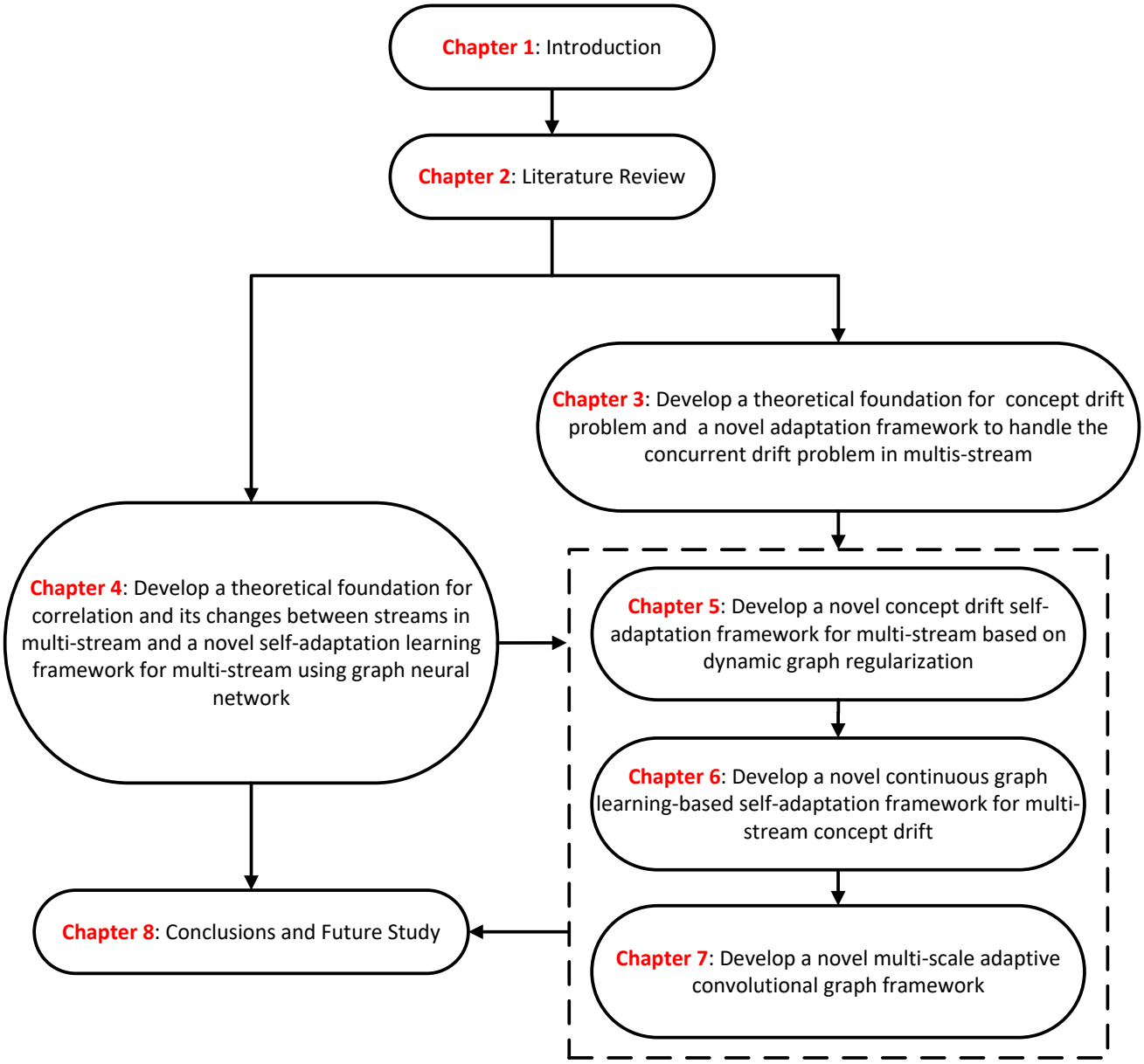


Figure 1.1: Thesis structure

CHAPTER 2

LITERATURE REVIEW

In this chapter, we review recent papers related to this thesis, which includes two main parts: concept drift and graph neural networks modelling for multi-stream.

2.1 Concept Drift

Concept drift refers to the phenomenon where statistical variables or data distributions in an unstable data stream change unpredictably over time [10]. When the relationship between newly generated data and historical data weakens significantly, traditional inductive methods designed for stationary data environments often fail to adapt, leading to inaccurate or even completely misleading predictions that deviate from actual trends [11].

Over the past decade, the study of concept drift has gained increasing attention, with numerous drift detection and adaptive learning algorithms being developed for various applications. However, despite the growing interest in this topic, comprehensive studies on concept drift handling techniques remain limited. This is partly due to the broad

scope of the concept drift problem, which often spans multiple research fields, making it a complex and interdisciplinary challenge.

2.1.1 Concept Drift Detection

Drift detection plays a crucial role in addressing the concept drift problem. This process encompasses techniques and mechanisms aimed at identifying and quantifying concept drift by detecting change points or time intervals where drifts emerge. This section provides an overview of drift detection methods and algorithms, classifying them into two primary categories based on the types of test statistics they employ.

2.1.1.1 Error Rate-based Drift Detection

Error-rate-based drift detection is a widely used strategy for identifying concept drift. This approach monitors changes in the online error rate of the model. When the accumulated error exceeds a pre-defined threshold, it signifies a statistical deviation, triggering a drift alarm and initiating the corresponding model update and adaptation process.

The Drift Detection Method (DDM) [20] is one of the most frequently cited algorithms in this category. Its core idea relies on the concept of context - a set of consecutive samples with a fixed distribution where training examples are presented sequentially. DDM determines whether the distribution has shifted by detecting significant increases in the online error rate within a time window. It is the first algorithm to define "warning" and "drift" levels for drift detection. Building on this foundation, the Early Drift Detection Method (EDDM) [21] modifies the calculation of test statistics, using the distance between consecutive correct classifications to estimate distribution changes. Another extension, the Statistical Test of Equal Proportions Detector (STEPD) [22], applies statistical tests of equal proportions to refine drift detection, enhancing the precision of the detection process compared to its predecessors.

Several algorithms further refine drift detection by improving sensitivity to localized or gradual changes in the instance space. Learning with Local Drift Detection (LLDD) [23] focuses on detecting drift within localized regions of the instance space. By associating drift detection problems with decision tree nodes, LLDD allows for monitoring and adaptation in specific areas, enabling a faster response to localized changes. New Drift Detection Method for Data Streams (NDDM) [24] takes a batch-based approach, comparing consecutive data batches to identify drift. For each instance in the current batch, it calculates differences based on nearest neighbors in the previous batch. Similarly, EWMA for Concept Drift Detection (ECDD) [25] employs an exponentially weighted moving average (EWMA) chart to monitor misclassification rates. ECDD's modular design enables it to work alongside any base classifier, offering a lightweight, fully online solution with low computational cost.

Some algorithms address specific challenges, such as long-term concepts or overlapping data distributions. The Reactive Drift Detection Method (RDDM) [26] is designed to handle long-term concepts by discarding older examples. Algorithms like Hoeffding's Bounds-based Drift Detection Method (HDDM) [27] and its extension, the Fast Hoeffding Drift Detection Method (FHDDM) [28], incorporate Hoeffding's inequality to evaluate statistical significance. These methods are particularly effective in capturing abrupt changes in data streams. The Fuzzy Windowing Drift Detection Method (FW-DDM) [29] introduces fuzzy logic to drift detection, using overlapping windows to improve accuracy in detecting overlapping or ambiguous concepts. In contrast, Dynamic Extreme Learning Machine (DELM) [30] combines drift detection with an adaptive neural network structure. DELM uses an online learning mechanism to train an Extreme Learning Machine (ELM) [31], adding hidden nodes upon detecting drift to enhance generalization. When drift occurs, DELM replaces the old classifier with a newly trained one, ensuring adaptability to the new concept.

2.1.1.2 Data Distribution-based Drift Detection

Drift detection based on data distribution is a widely adopted strategy for identifying concept drift. This approach measures differences between new and historical data distributions, often using window-based mechanisms as the primary carrier. Typically, algorithms employ two fixed historical windows alongside a sliding window for new data [32–34]. A distance metric quantifies the distributional differences, and when these exceed a statistical threshold, the learning model is updated accordingly. However, this approach is computationally intensive compared to error-rate-based methods due to the additional calculations required [35].

A seminal contribution in this area is by [36], which introduced practical distribution-based drift detection. The study highlighted total variation as a natural distance metric between distributions, expressed as $TV(P_1, P_2) = 2\sup_{E \in \varepsilon} |P_1(E) - P_2(E)|$ or equivalently in terms of density functions as $dist_{L^1} = \int |f_1(x) - f_2(x)| dx$. Another influential algorithm is ADWIN [37], which dynamically adjusts its sliding window size based on detected drift. ADWIN maintains two sub-windows representing old and new data and calculates their optimal division by assessing distributional differences. If the difference between the average values of these sub-windows exceeds a threshold, drift is declared, and the new data retrains the model. ADWIN has inspired numerous adaptations [37–41], extending its applicability across diverse scenarios.

Other algorithms leverage specific statistical measures to enhance drift detection. [33] proposed a method using relative entropy (KL-divergence) to quantify distributional differences without parametric assumptions. This algorithm also integrates spatial scanning statistics, enabling the identification of regions with significant changes. Similarly, SCD [42] employs density tests for multidimensional data, while CM [35] combines noise filtering and redundancy removal to improve detection precision.

Some approaches further explore density-based techniques. LSDD-CDT [43] applies

minimum square density difference estimation for online, multidimensional input. Its incremental version, LSDD-INC [44], compares non-overlapping data windows to measure distributional stationarity. EDE [45] estimates iso-density regions for non-parametric drift detection in unstable spaces, while LDD-DSDA [46] monitors local density changes to measure regional drift.

Multiple hypothesis testing methods offer a distinct approach by embedding parallel or hierarchical tests within the detection process. These algorithms enhance sensitivity and scalability for adaptive concept drift detection [47–52].

Concept drift detection methods each with distinct strengths and limitations. Error-rate-based methods are efficient and straightforward, relying on tracking changes in model performance over time. Their low computational cost makes them suitable for real-time applications, but they often struggle to provide precise drift localization or adapt to gradual changes effectively. On the other hand, distribution-based methods excel at identifying the root causes of drift by analyzing data distribution changes. These methods can pinpoint drift locations and are particularly effective for complex, multidimensional data streams. However, they come with higher computational costs and may require careful parameter tuning, such as window size, to balance sensitivity and stability. Both approaches play complementary roles in addressing the diverse challenges posed by concept drift in real-world scenarios.

2.1.2 Concept Drift Adaptation

Concept drift leads to changes in data distribution, requiring model adjustments to maintain prediction accuracy. The previous section discussed strategies and methods for drift detection, which are closely linked to model adaptation. Adaptation frameworks typically follow two main approaches: informed vs. blind [53], or active vs. passive [11]. Despite the terminology differences, the core distinction lies in whether a drift detection

mechanism initiates the adaptation process. Active approaches employ a detection mechanism to trigger adaptation based on specific feedback, whereas passive approaches continuously update the model with the data stream, regardless of whether drift is detected.

2.1.2.1 Approaches for Active Adaptation

Active approaches integrate concept drift detection mechanisms into the algorithm, triggering adaptation processes when drift is identified. Sliding windows are a classic strategy for this purpose, typically holding the latest available data. Upon detecting drift, the classifier or prediction model is retrained using the new data in the window, while old samples are discarded to adapt to incoming patterns. However, fixed window lengths often fail to accommodate diverse data stream types.

ADWIN [37] determines the optimal subwindow size based on the rate of change in the difference between two subwindows' data distribution characteristics. This approach eliminates the need for predefined window sizes, improving generalization capabilities. Expanding specific machine learning algorithms is another common drift adaptation technique. For instance, DELM [30] extends the Extreme Learning Machine (ELM) framework [31] by adding hidden layer nodes when drift is detected, enhancing the model's fitting ability. FP-ELM [54] introduces forgetting parameters into ELM, enabling drift adaptation. SAGA [55] employs a region-based filtering method to update outdated models, while other methods define drift gradients over training segments [56] to quantify distribution changes and distinguish drift instances from noise.

JIT (Just-In-Time) adaptive classifiers [47, 57, 58] refine window sizes using confidence interval intersections, often combining distribution and error rate tests to detect drift. Hybrid methods like IOLIN [8] alternate between fixed-length windows during stable periods and adaptive mechanisms during drift, saving computational resources with dynamic adjustments.

Bias weighting provides another adaptive strategy. Unlike sliding windows, weighting assigns importance to all samples, often using time-based forgetting functions [59]. These functions reduce the influence of older data, with adaptations including attenuation-based mechanisms [60] and change indices for weighting based on data generation dynamics [61]. While effective, weighting can be computationally expensive and memory-intensive, making it less suitable for big data scenarios.

Sampling is also widely applied. Reservoir sampling [62] enables random selection of elements from data streams without replacement, as seen in adaptive strategies for evolving river data [63]. This approach balances computational efficiency with representational adequacy.

2.1.2.2 Approaches for Passive Adaptation

Passive approaches, as the name suggests, do not incorporate a specific drift detection mechanism within the algorithm. However, this does not imply that the model remains static. Each new data point presents an opportunity for adjusting the model's parameters, enabling continuous adaptation to potential future changes.

A common passive strategy involves updating a single classifier, which is more efficient than retraining since it adapts only relevant parts of the model. The decision tree algorithm is a well-known framework for this approach. Very Fast Decision Tree (VFDT) [64], an enhancement of the Hoeffding tree, uses Hoeffding's inequality to identify optimal attributes for decision nodes in a decision tree. However, VFDT assumes smooth data distribution, which limits its ability to adapt to concept drift. The Concept-adapting Very Fast Decision Tree (CVFDT) [65] overcomes this by introducing a sliding window, allowing for continuous updates and ensuring model accuracy in the presence of drift. VFDTc [66] further extends VFDT by handling continuous data and utilizing more powerful classification techniques. The IADEM-3 model [67], based on the decision tree approach, addresses issues with Hoeffding's boundary by using sums of independent

random variables for split calculations.

In addition to local updates of a single classifier, ensemble methods have gained significant attention in streaming data mining, especially for recurring concept drift. Retaining old models can prevent unnecessary retraining and reduce error variance, often resulting in higher accuracy compared to single classifier systems. These methods automatically incorporate new data while using forgetting mechanisms to remove outdated models [68, 69]. The Stream Integration Algorithm (SEA) [70], for instance, builds a separate classifier for each new batch of data and combines them into a fixed-size set with a heuristic replacement strategy, ensuring efficient memory usage and swift adaptation to drift.

Classic ensemble strategies, such as Bagging, Boosting, and Random Forests, have also been adapted for streaming data with concept drift. Online versions of bagging and boosting algorithms [71] are trained incrementally with each instance to simulate batch learning. The leverage Bagging algorithm [72] improves upon this by integrating ADWIN to manage concept drift. Upon detecting drift, a new classifier is trained, replacing the outdated model.

Ensemble methods have also evolved with novel voting techniques for drift adaptation. The Dynamic Weighted Majority (DWM) [73] algorithm, based on the weighted majority algorithm (WM) [74], uses weighted voting rules to manage drift. When a classifier misclassifies an instance, its weight decreases, and if the weight drops below a threshold, the classifier is removed from the ensemble. Other adaptive methods, such as Dynamic Ensemble of Ensembles (DE2) [75] and Ensemble-tree (E-tree) [76], use similar strategies to adjust ensemble sizes dynamically. Methods like [77] and [78] also adjust ensemble size to respond to concept drift.

Limitations: The above methods provide rich references for developing concept drift adaptation frameworks, but most of them ignore the case of multi-stream environments.

Although methods applicable to single streams can be applied repeatedly to multiple streams, this approach is rough and inefficient. The correlations between streams will be ignored, and isolated modelling cannot guarantee the synchronicity of tasks across multiple streams. Little research has been dedicated to addressing this gap. The variance matrix is employed to measure the similarity between two streams in [79]. While these works provide motivation for this area, further improvements are necessary to overcome their limitations and deficiencies.

2.2 Graph Neural Networks Modelling for Multi-stream

Graph Neural Networks (GNNs) have emerged as an effective tool for capturing deep relationships within complex networks, even for non-graph structured data, by constructing correlation graphs in a non-Euclidean space based on distance or other metrics [80, 81]. They can handle data with irregular structures and capture high-order dependencies, making them particularly suitable for scenarios where traditional methods may struggle [82, 83]. Furthermore, GNN-based approaches have demonstrated impressive performance in various applications, including traffic prediction tasks [84, 85]. These methods have achieved notable results by leveraging the rich relational information encoded in graphs, showcasing their potential for multi-series prediction tasks and beyond.

In popular traffic prediction tasks, the most common approach for multi-stream correlation construction is to initialize the graph structure based on pre-defined graphs and then utilize GNNs to capture high-order spatio-temporal relationships. Many effective frameworks have been proposed. Typically, the distance between sensors is used as the metric for evaluating similarity. DCRNN [86] is a method proposed for traffic streaming

sequence data. The model builds a diffuse convolutional network on a graph structure. The bidirectional walk and predetermined sampling capture the spatial and temporal dependencies of multiple streams respectively. STGCN [87] combines the graph convolutional sequence learning layers and graph convolutional layers to extract the spatial and temporal dependencies features within streams. Graph-WaveNet [88] calculate an efficient adjacency matrix on the pre-defined graph, by combining graph convolution and dilated casual convolution to extract dependencies information of nodes at different granular levels. STFGNN [89] emphasizes the construction of temporal graphs to enhance prediction performance. ASTGCN [90] introduces a latent network composed of a Laplace matrix to dynamically represent spatiotemporal connections within traffic data. DGCRN [91] designs filters with learnable parameters to capture historical features and generate dynamic graphs during training.

Pre-training is also a strategy to enhance model performance. GMAN [92] proposes a graph multi-attention network embedded in a pre-training wor2vec matrix that shows the Performance improvements. In [93], the authors build a pre-training model using transformer techniques to achieve better accuracy. Likewise, in [94], a pre-training model based on a transformer module is employed. The designed estimation gate and a residual decomposition mechanism assist the model in separating diffusion and inherent traffic information in a data-driven manner. However, the requirement for pre-defined graphs is too stringent for many practical applications. The assumption that a predefined graph can always be obtained makes research in this area highly limiting.

Several researchers are trying to develop frameworks without using pre-defined graphs. [95] proposes adaptive graph modules to represent dependencies between streams from historical features using learnable parameters. The sampling method is introduced to the graph construction process in [96], and the simulated probability distribution graph is parameterized by a neural network. These methods achieve good

prediction performance without using pre-defined graphs.

Limitations: While these methods introduce GNNs into non-graph structured data and leverage spatio-temporal correlation graphs to enhance model performance, they largely overlook the concept drift problem in data streams. Data streams generated by sensors exhibit rapid changes, making concept drift inevitable. For example, the sudden traffic accidents or extremely bad weather in practical application scenarios. The offline prediction mode just based on the historical data distribution is too limited. Therefore, the effectiveness of these frameworks can only be validated on consistent data distributions. When the data distribution changes, relying solely on historical data will not be sufficient to meet actual prediction demands.

CHAPTER 3

AN EFFICIENT BAYESIAN NEURAL NETWORK FOR MULTIPLE DATA STREAMS

3.1 Introduction

Streaming data is widespread in many practical scenarios such as finance, transportation, weather, social, etc., where infinite data instances arrive in a sequential way [4]. The streaming data mining has been one of the most important research topics in recent years[5], especially how to overcome the intrinsic uncertainties and obtain accurate real-time predictions have attracted increasing attention in this area [6]. One of the most common uncertainties in data streams is the occurrence of concept drift [53]. Concept drift refers to the phenomenon that data distributions change over time[3]. For example, the weather forecast of 'rain/no rain' largely depends on whether it is in the dry season or wet season [97].

In conventional machine learning methods, the training set and test set are assumed to have the same probability distribution [7], so that a predictor trained with the training

set can still be used to predict the testing set. Once concept drift occurs, the trained predictor will be probably invalid to predict the testing data because they have different or even opposite data patterns, resulting in a decrease in the prediction accuracy of the predictor for the coming data [46, 98]. In data stream prediction, when concept drift occurs frequently, the predictor is not applicable if it is trained with only fixed historical data [99].

Current research shows that concept drift handling techniques can solve this problem effectively [100]. The detection, the quantification and the adaptation are currently challenging problems in the field of concept drift [12]. Drift detection is to detect when the concept drift occurs [11] by monitoring the learner error [13] or a designed statistic [14]. Regarding the quantification of conceptual drift, it involves retrieving the time when the drift occurred, the severity of the drift, and the area of the drift. Most of the drift detection algorithms can measure time and severity of the drift, but whether the specific location of the drift can be determined depends very much on the nature of the predictor itself [15]. The purpose of concept drift adaptation is to integrate the latest data machine statistical distribution information into the predictor to maintain or improve the prediction accuracy. There are usually two strategies to adjust the predictor [101]. The first is to retrain the predictor with the latest data to adapt to the new distribution, and the second is to update the original predictor with new data training [16, 17].

So far, the concept drift handling techniques have been validated in many single data stream cases [4, 18]. However, handling multiple data streams with concept drift is still an unsolved question. In addition, data streams in real scenarios basically do not exist independently, and data mining on multiple related data streams is the real dilemma [19]. Taking a traffic scene as an example, the bus routes of different lines in a city may overlap or intersect. The operating information of these lines is concurrent in time, and there is a certain degree of dependency between each other.

To fill the research gap in multiple streams, this chapter proposes a concept drift adaptation method called $MuNet$ for multiple dependent data streams. Especially, we discuss the scenario that multiple data streams contain the concurrent drift, that is, the similar concept drift would occur simultaneously for all these data streams. Taking a traffic scenario as an example, when there is traffic congestion on overlapping lines, all buses on the same road section will be affected simultaneously. It is also easy to be extended to other scenarios for application, such as the monitoring of water quality and water level in a certain water area, the speed prediction of each wind turbine in a certain wind power station, etc.

In $MuNet$, we propose a novel adaptation strategy for multiple data streams with concurrent drift and we use Bayesian neural networks [102, 103] to implement this strategy. The main idea is using updated predictors to make adaptation on *only one* stream and using connectors to make adaptation for *the other* streams. As it takes less time to update the connector than updating the predictor, the computational cost has been largely decreased when predicting multiple data streams simultaneously. In this chapter, we use a single-stream adaptation method to update predictors for the base stream, and design a Bayesian neural network as the connector, named *BNN-connector*. These connectors can continuously learn the drift information from the base stream and use the learned information to correct the prediction results for other streams.

The work described in this chapter has been published in the IJCNN 2021 conference paper "An Efficient Bayesian Neural Network for Multiple Data Streams" .

3.2 Problem Statement

Before we explain the proposed method in details, related notations and definitions are listed in this subsection.

Notations:

- t: $t \in \mathbb{Z}^+$ presents the time point
- τ: $\tau > 1$ presents a time period
- $\mathscr{P}$: probability distribution
- p: the probability for discrete cases or probability density function for continuous cases
- $\mathrm{X_t} = (X_t^1, ..., X_t^m)$: time series for features or attributes
- $\mathrm{y_t} = (y_t^1, ..., y_t^l)$: time series for labels. in this chapter, we only consider l=1
- $s = \{(\mathrm{X_t}, \mathrm{y_t})\}$: data stream

Definition 3.1 (Single Data Stream)**.** A data stream $D_t = \{(\mathrm{X}_t, y_t) | t = 1, ...\infty\}$, generated from distribution $\mathscr{P}_t$ with $p_t(\mathrm{X}, y)$ its probability function or probability density function (pdf), is received, where $\{\mathrm{X}_t \in \mathbb{R}^d\}$ is the attribute variable (or the input) consisting of d time series, for some d, and $\{y_t \in \mathbb{R}^1\}$ is the label variable (or the scalar output).

Definition 3.2 (Concept Drift in Single Data Stream [55])**.** Concept drift occurs in a data stream if $\exists t_{d^i}$ that

$$\begin{cases} p_{t+1}(\mathrm{X}, y) \neq p_t(\mathrm{X}, y), \text{ for } t = t_{d^{(i)}} \\ p_{t+1}(\mathrm{X}, y) = p_t(\mathrm{X}, y), \text{ for } t \in \left(t_{d^{(i)}+\tau_i}, t_{d^{(i+1)}}\right) \end{cases} \tag{3.1}$$

where $\forall i,\ t_{d^{(i+1)}} - t_{d^{(i)}} > 1$, $t \in \mathbb{Z}^+$ presents the time step, $d^{(i)}$ is an order statistics denoting the i^{th} drifted time point, and $1 < \tau_i < t_{d^{(i+1)}} - t_{d^{(i)}}$ is specifically for the occurrence of incremental drift.

Remark 1. *A data stream contains concept drift if the data pattern changes at least once, namely* $\{t_{d^{(i)}}\} \neq \emptyset$ *that* $p_{t+1}(\mathrm{X}, y) \neq p_t(\mathrm{X}, y)$*, for* $t = t_{d^{(i)}}$*; in addition, the changed pattern is not ephemeral, but will last for a period (at least last for two time steps), which is manifested by* $\forall i,\ t_{d^{(i+1)}} - t_{d^{(i)}} > 1$*. The pattern stays the same in this period*

that $p_{t+1}(\mathrm{X}, y) = p_t(\mathrm{X}, y)$*, for* $t \in \left(t_{d^{(i)}+\tau_i}, t_{d^{(i+1)}}\right)$*; here* $\tau_i = 1$ *when the drift occurs suddenly while* $\tau_i > 1$ *when the drift occurs incrementally in the period of* $\left(t_{d^{(i)}+1}, t_{d^{(i)}+\tau_i}\right)$.

Definition 3.3 (Learning Aim at t-step for Single Data Stream)**.** To predict the value of the label variable for a data stream at time step t, the learning aim is to obtain a predictor H_t for $p_t(\mathrm{X}, y)$, which can be denoted as

$$H_t = \underset{h \in \mathscr{H}}{\arg\min}\ \ell(h, \mathrm{X}, y | (\mathrm{X}, y) \in p_t(\mathrm{X}, y)), \tag{3.2}$$

where $\mathscr{H}$ is the hypothesis set, $\ell : \mathbb{R}^1 \times \mathbb{R}^1 \to \mathbb{R}_+$ is the loss function used to measure the magnitude of error.

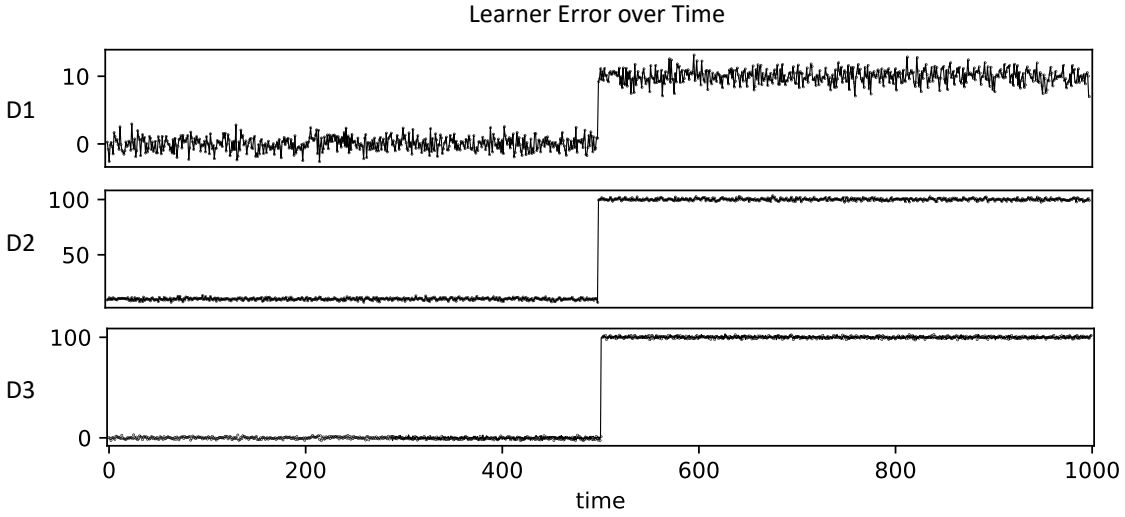


Figure 3.1: An example of concurrent drift among data streams. It can be inferred that data streams D^1, D^2 and D^3 have a sudden error increase (sudden drift) at the same time when $t = 500$.

Definition 3.4 (Concurrent Drift in Multiple Data Streams)**.** Given D_t^u, D_t^v (where $u \neq v$) two different data streams, p^u is the probability or probability density function for stream u while p^v is that for stream v. A *concurrent drift* occurs if $\exists t_{d^{(u)}}$ that

$$\begin{cases} p_{t+1}^u \neq p_t^u, p_{t+1}^v \neq p_t^v \text{ for } t = t_{d^{(i)}} \\ p_{t+1}^u = p_t^u, p_{t+1}^v = p_t^v \text{ for } t \in \left(t_{d^{(i)}+\tau_i}, t_{d^{(i+1)}}\right) \end{cases} \tag{3.3}$$

Remark 2. *It can be seen that concurrent drift occurs when concept drift occurs in two data streams at the same time. It should be noticed that the concurrent drift could occur among more than two data streams, such as is shown in Fig. 3.1. In Fig. 3.1,* D^1*,* D^2 *and* D^3 *represent three data streams separately and each sub-figure shows the learner error of a static predictor changes over time. Normally, error is the most common measurement to determine whether drift occurs [79]. Clearly, concurrent drift occurs among* D^1*,* D^2*, and* D^3 *as drift occurs at* $t = 500$ *for* D^1*,* D^2*, and* D^3*.*

Definition 3.5 (Learning Aim at t-step for Multiple Data Streams)**.** For multiple data streams $D^1, D^2, \cdots, D^S$ at time step t, the learning aim is to obtain a set of predictors $H_t^1, H_t^2, \cdots, H_t^S$ for $p_t\left(D^1, D^2, \cdots, D^S\right)$, which can be denoted as

$$H_t^s = \underset{h^s \in \mathscr{H}^s}{\arg\min}\ \ell\left(h^s, D^s \middle| D^1, \cdots, D^S \in p_t\left(D^1, \cdots, D^S\right)\right), \tag{3.4}$$

where $\mathscr{H}^s$ is the hypothesis set for D^s and $s \in \{1, \cdots, S\}$.

Remark 3. *According to Definition 3.5, the learning aim in a multi-stream scenario is to obtain multiple predictors at each time point with considering the relationship among streams. If we consider each stream separately, Definition 3.5 will be degenerated to the following case*

$$H_t^s = \underset{h^s \in \mathscr{H}^s}{\arg\min}\ \ell\left(h^s, D^s \middle| D^s \in p_t\left(D^s\right)\right). \tag{3.5}$$

3.3 Methodology

Given the learning aim in Definition 3.5, the next step is how to obtain the optimal H_t^s. If the data streams do not have concept drift problem and the streams are independent with each other, the probability function stays the same and $H_t^s \equiv H^s$. That means we could use a period of historical data to train a predictor H^s and use this predictor to predict without retraining or tuning H^s. However, due to the occurrence of concept drift,

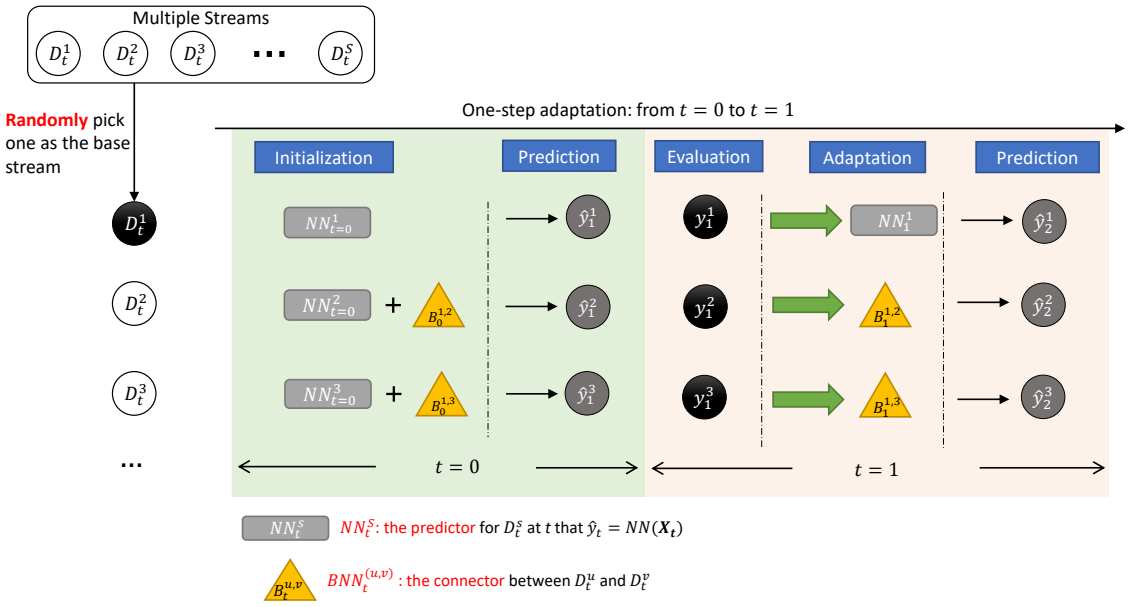


Figure 3.2: The One-step Adaptation Procedure for Multiple Streams with Concurrent Drift.

the assumption that data streams have unchanged data distributions has been broken. In the meantime, data streams are assumed to be dependent with each others in the multi-stream scenario.

Concept drift is a common phenomenon in data streams and it is unknown in advance whether drift occurs in a data stream. It is risky to use stationary model for an arbitrary data stream as the stationary model will lead to terrible prediction results when drift occurs while drift-adaptive models are still suitable to predict data streams that do not contain concept drift. Therefore, the design of connector is very useful for many real-world applications related to multiple data streams.

In this chapter, we consider the case when streams have concurrent drift (Definition 3.4). We randomly pick one stream as the base stream. We train and update the predictor for the base stream to handle the drift problem. As for the other streams, we only train an initial stationary predictor with a batch of historical data. After that, predictions from the initial predictor will be *corrected* by an online-learned *connector*. Compared to the

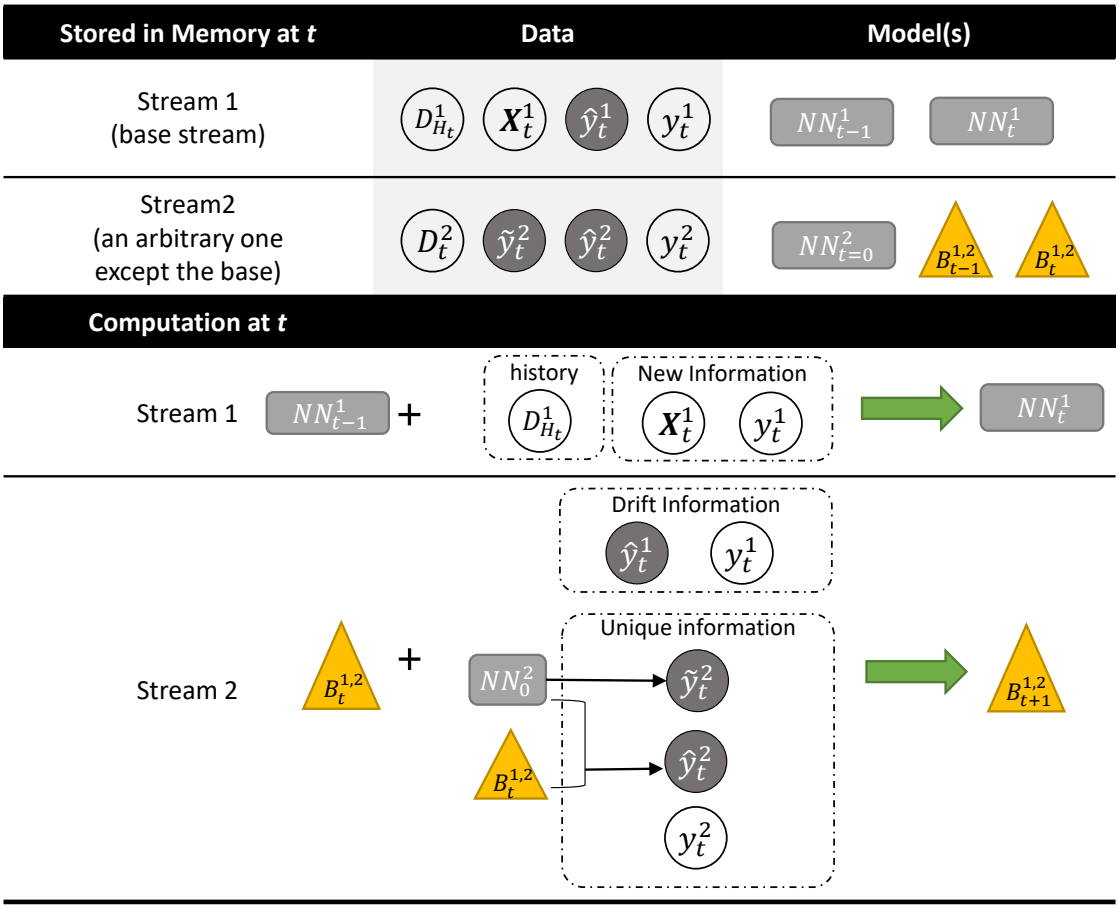


Figure 3.3: The proposed $MuNet$.

predictor (a complex neural network), the connector (a neural network with 1-2 layers) takes much less computational cost. By this way, $MuNet$ can efficiently decrease the computational cost when predicting multi-streams task.

For better understanding, we first introduce the general idea of how we make drift adaptation for multiple streams with concurrent drift. After that, we give technical details of how $MuNet$ implements that adaptation idea.

- The one-step adaptation for multiple streams with concurrent drift.

The one-step adaptation procedure for multiple streams with concurrent drift is presented in Fig. 3.2. There are S data streams and we assume that these streams have

concurrent drift. Current drift adaptation methods are designed for single data stream. Therefore, if we directly use those methods to solve multi-stream task, the adaptation for all streams will be the same as the adaptation for D_t^1 in Fig. 3.2. That is to retrain the predictor periodically, for example retraining every batch of new arrived instances (where t denotes a time period) or even every one instance (where t denotes a time point).

Although retraining predictors is an efficient way to solve the problem of concept drift in data streams, it often has high computational cost. Considering the unique characteristics of concurrent drift, we assume that the drift pattern of one data stream can be learned and applied to other streams because these streams have drift at the same time. Therefore, we propose to *update predictors* for one base data stream by retraining and *correct prediction results* for other streams by updating connector built between the base stream and others.

For example, in Fig. 3.2, we first pick one stream from S streams as the base stream. It should be noticed that in this chapter we pick the base stream randomly and we also give an analysis of accuracy fluctuate with selecting different base streams in Section experiment. At $t = 0$, the predictors and connectors are *initialized*. In this chapter, we use a complex neural network as the predictor and a two-layer Bayesian neural network as the connector. With the initialized predictors and connectors, we can *estimate the labels* for $t = 1$ which are denoted by $\hat{y}_1^1$ for stream D^1, $\hat{y}_1^2$ for stream D^2, and so on.

As discussed above, we have the initialized predictors and connectors, as well as the prediction results at $t = 0$. Then the time goes to $t = 1$, we obtain the true label y_1^1, y_1^2, and so on. At $t = 1$, we evaluate the predictions of $\hat{y}_1^1, \cdots, \hat{y}_1^S$ and then use the true labels to update the predictor on the base stream and connectors on other streams. The updated predictor and connectors are used to predict labels for $t = 2$. Then the time goes to $t = 2$, we repeat the evaluation, adaptation and prediction processes. Unlike the stationary machine learning methods, the real-time prediction uses this prequential evaluation

method [53].

- Handling concurrent drift with $MuNet$.

So far, we have discussed the main idea of updating predictors and connectors separately to reduce the computational cost. Next, we will explain how this idea is used in our proposed $MuNet$.

We introduce our $MuNet$ from both aspects of memory and computation at each t. Compared to Fig. 3.2 which includes general processes in both $t = 0$ and $t = 1$, Fig. 3.3 presents details at $t = 1$. As this procedure is repeated for each t, the case $t = 1$ is also the case for an arbitrary t that $t \neq 0$.

As is shown in Fig. 3.2, $MuNet$ only stores the most current data. For example, if we only have two streams D_t^1 and D_t^2, and D_t^1 is chosed as the base stream, the stored data for base stream at t are a batch of historical data $D_{H_t}^1$, the predicted value($\hat{y}_t^1$), and the newly arrived data(X_t^1 and y_t^1). Compared to other streams, we store an extra batch of data with fixed length for the base stream, and this batch of data will be updated by including the newest data and deleting the oldest data at each t.

The stored data for other streams at t are the predicted values($\hat{y}_t^2$, $\tilde{y}_t^2$) and the newly arrived data(X_t^2 and y_t^2). Compared to the base stream, there are two predicted values for Stream 2 where $\tilde{y}_t^2$) is the predicted value by the initial predictor and $\hat{y}_t^2$ is the predicted value by the initial predictor corrected by the connector $BNN_t^{1,2}$.

With these stored values, $MuNet$ is self-adapted as is shown in the computation part in Fig. 3.2. The adaptation of $MuNet$ on Stream 1 is to retrain the neural network predictor NN_{t-1}^1 with the historical information and the new information. The adaptation of $MuNet$ on Stream 1 is to tune the Bayesian neural network with drift information from base stream and unique information for Stream 2.

When the predictor has high computational cost for updating and the connector cost low computational cost for updating, $MuNet$ will efficiently decrease the total

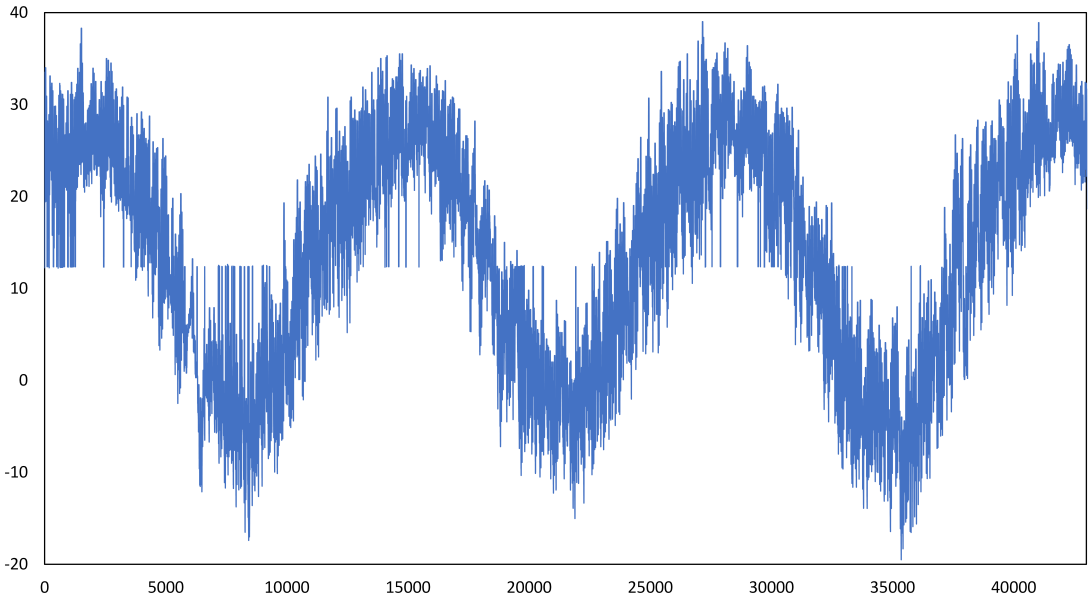


Figure 3.4: The change of target variable *t2m_obs_gt* historical data.

computational cost, especially there are many streams to predict. For example, in weather prediction, we need to know the prediction for all weather station at the same time. These weather stations are normally built not far away from each other in one area. Therefore, the data streams from these stations have concurrent drift.

3.4 Experiments

We validate the proposed $MuNet$ by a real-world dataset of weather prediction. In this section, we will first introduce the data in Section 3.4.1. Section para lists the parameters used in $MuNet$ and the experimental results are presented in Section results.

3.4.1 Data Description

We use a public dataset collected from the weather stations in Beijing, China. This data contains hourly weather data from ten weather stations during time zone (3:00 intraday

to 15:00 of the next day) from 03/01/2015-05/31/2018 (1188 days in total) where missing values were deleted during the testing. Each station contains same features.

To mimic the real prediction scenario, the original data has been pre-processed by overlapping in [104]. In this chapter, we use the same pre-processed data as is in [104]. In our experiment, we use the data from ten stations. Each station data contains nine attributes, including *psur_obs* (ground pressure), *t2m_obs* (temperature at 2 meters), *q2m_obs* (specific humidity at 2 meters), *w10m_obs* (wind speed at 10 meters), *d10m_obs* (meridional wind at 10 meters), *rh2m_obs* (relative humidity at 2 meters), *u10m_obs* (warp wind direction at 10 meters), *v10m_obs* (zonal wind at 10 meters), *RAIN_obs* (accumulated rainfall in one hour at ground). The target variable is *t2m_obs_gt* (temperature at 2 meters). *"_obs"* represents the observed value, *"_obs_gt"* represents the ground truth value.

Clearly, each station contains 10 times series as one data stream, and we have 10 data streams. These data streams have concept drift due to the intrinsic characteristics of atmospheric variables. Take the target variable *t2m_obs_gt* as an example, it has obvious seasonal changes according to the three-year data record, as shown in Fig. 3.4. As these stations are located near to each other, the corresponding data streams have the concurrent drift.

Table 3.1: The performance comparison of stationary predictor and adaptive predictor

	S_i_*Stationary*	S_i_*Adaptation*
S_0_*MAPE_AVG*	10.67%	6.57%
S_1_*MAPE_AVG*	10.41%	6.32%
S_2_*MAPE_AVG*	10.63%	6.66%
S_3_*MAPE_AVG*	10.76%	6.69%
S_4_*MAPE_AVG*	10.32%	6.31%
S_5_*MAPE_AVG*	9.98%	6.34%
S_6_*MAPE_AVG*	11.15%	7.23%
S_7_*MAPE_AVG*	10.26%	6.25%
S_8_*MAPE_AVG*	10.29%	6.23%
S_9_*MAPE_AVG*	10.52%	6.80%

3.4.2 Experimental Settings

The experiments were implemented on a CPU server with Core i7 7700K CPU and Pytorch (1.8.1) programming environment.

The $MuNet$ contains *predictors* and *connectors*. The *predictors* for all data streams uses a 5-layer neural network and each layer contains 128 neurons. The input of the predictor is the 9 meteorological attribute observations at the observation point at time t, and the output is the predicted value of the target variable at time $t+1$. The training epoch of predictors is 2000 for all streams. All predictors use the stochastic gradient descent (SGD) method, and the learning rate is 0.01. As for the predictors initialization, we use 2400 hours of recorded weather data in the same time period of all streams as the training data for their initial predictors. The predictor of base stream is retrained every 120 hours by including the newly arrived 120 hours' data and deleting the oldest 120 hours' data.

The *BNN-connector* is a Bayesian neural network containing 2-layer hidden layers and each layer contains 30 and 10 neurons respectively. The training epoch of each *BNN-connector* is 30. All *BNN-connectors* use the SGD method, and the learning rate is $1e-1$. *BNN-connectors* are updated by continuously tuned by the newly arrived instances.

Table 3.2: The performance comparison of the MuNet and conventional adaptation method

	S_i_Adaptation	$MuNet$ (S_i_base)									
		S_0_base	S_1_base	S_2_base	S_3_base	S_4_base	S_5_base	S_6_base	S_7_base	S_8_base	S_9_base
S_0_MAPE_AVG	6.57%	**6.57%**	7.24%	7.36%	7.29%	7.28%	7.33%	7.41%	7.45%	7.18%	7.28%
S_1_MAPE_AVG	6.32%	6.94%	**6.32%**	7.16%	6.99%	6.88%	6.95%	7.38%	7.11%	6.97%	6.98%
S_2_MAPE_AVG	6.65%	7.39%	7.25%	**6.66%**	7.48%	7.25%	7.72%	7.53%	7.39%	7.21%	7.34%
S_3_MAPE_AVG	6.68%	7.36%	7.36%	7.49%	**6.69%**	7.37%	7.50%	7.63%	7.56%	7.44%	7.40%
S_4_MAPE_AVG	6.30%	6.88%	6.85%	6.99%	7.00%	**6.31%**	6.98%	7.24%	7.01%	6.86%	6.89%
S_5_MAPE_AVG	6.33%	6.96%	6.86%	7.03%	7.00%	6.93%	**6.34%**	7.14%	7.04%	6.83%	7.00%
S_6_MAPE_AVG	7.22%	8.16%	7.97%	8.30%	8.21%	8.02%	8.56%	**7.23%**	8.12%	7.95%	8.53%
S_7_MAPE_AVG	6.27%	6.67%	6.64%	6.68%	6.70%	6.65%	6.65%	6.76%	**6.25%**	6.64%	6.62%
S_8_MAPE_AVG	6.21%	6.91%	6.74%	7.12%	6.83%	6.77%	6.83%	6.96%	6.96%	**6.23%**	6.83%
S_9_MAPE_AVG	6.79%	7.49%	7.38%	7.65%	7.44%	7.38%	7.45%	7.61%	7.65%	7.44%	**6.80%**
Time Cost	886	350	361	373	358	347	355	370	354	350	360

Table 3.3: The computational cost of different methods as the number of data streams increases (S_0 as the base stream)

	One Streams		Two Streams		Three Streams		Ten Streams	
	$S_i\_Adaptation$	$MuNet$	$S_i\_Adaptation$	$MuNet$	$S_i\_Adaptation$	$MuNet$	$S_i\_Adaptation$	$MuNet$
$S_0\_MAPE\_AVG$	6.57%	6.57%	6.57%	6.57%	6.57%	6.57%	6.57%	6.57%
$S_1\_MAPE\_AVG$	-	-	6.32%	6.94%	6.32%	6.94%	6.32%	6.94%
$S_2\_MAPE\_AVG$	-	-	-	-	6.65%	7.39%	6.65%	7.39%
$S_3\_MAPE\_AVG$	-	-	-	-	-	-	6.68%	7.36%
$S_4\_MAPE\_AVG$	-	-	-	-	-	-	6.30%	6.88%
$S_5\_MAPE\_AVG$	-	-	-	-	-	-	6.33%	6.96%
$S_6\_MAPE\_AVG$	-	-	-	-	-	-	7.22%	8.16%
$S_7\_MAPE\_AVG$	-	-	-	-	-	-	6.27%	6.67%
$S_8\_MAPE\_AVG$	-	-	-	-	-	-	6.21%	6.91%
$S_9\_MAPE\_AVG$	-	-	-	-	-	-	6.79%	7.49%
Time Cost (min)	87	**87**	167	**130**	248	**176**	886	**350**

3.4.3 Results

We have conducted three groups of experiments: a) comparisons between stationary methods and the adaptation methods; b) comparisons between the adaptation by the single data stream adaptation method and the adaptation by $MuNet$; c) time cost of $MuNet$ when adding extra streams. For all experiments, we use the mean absolute percentage error (*MAPE*) as the evaluation criterion. *MAPE* is calculated as follows:

$$MAPE = \frac{100\%}{n} \sum_{i=1}^{n} \left| \frac{\hat{y}_i - y_i}{y_i} \right|$$

Where $\hat{y}_i$ is the predicted value, y_i is the true value, and n is the number of samples. The smaller the value of *MAPE*, the closer the predicted value is to the true value. It means that the prediction model is more accurate. In the tables of experimental results, S_i*_MAPE_AVG* represents the average value of *MAPE* of all prediction results on the current data stream S_i, which can indicate the overall prediction level of the model.

- *Retraining predictors can solve the concept drift problem in data streams.*

The results of experiment a) are shown in Table 3.1 where S_i*_Stationary* uses an unchanged initial predictor while S_i*_Adaptation* retrains the predictor every 120 hours. It can be seen that the overall accuracy of S_i*_Adaptation* is much better than its corresponding accuracy of S_i*_Stationary*, indicating that these data streams contain concept drift problem. Experiment a) concludes that all these data streams contain concept drift problem, and retraining predictors is an efficient solution.

- *Compared to retraining, $MuNet$ provides comparable accuracy with less computational cost.*

Based on a), Experiment b) is to test whether we can use $MuNet$ to predict these multiple data streams with less computational cost and comparable accuracy. The results are listed in Table 3.2. In terms of accuracy, the accuracy of S_i*_Adaptation* is 6.57%,

6.32%, 6.65%, 6.68%, 6.30%, 6.33%, 7.22%, 6.27%, 6.21% and 6.79% for stream S_0 to S_9 separately. The accuracy of $MuNet$ with any base stream is a little lower than S_i_*Adaptation*. Take the accuracy of $MuNet$ when S_0 is selected as the base stream as an example, which is 6.57%, 6.94%, 7.39%, 7.36%, 6.88%, 6.96%, 8.16%, 6.67%, 6.91% and 7.49% for stream S_0 to S_9 separately. The accuracy of $MuNet$ is slightly lower than that of S_i_*Adaptation*, but it still maintains a good level.

From the perspective of time cost, $MuNet$ takes less time obviously. In addition, in order to prove that $MuNet$ can maintain this level when selecting any stream as the base stream, as shown in Table 3.2, we test the $MuNet$ performance by using ten data streams from S_0 to S_9 as the base stream separately. When using different base streams, the accuracy and time cost are slightly different but all the results are in a reasonable range.

- *$MuNet$'s advantage on time increases when the number of streams increases.*

We quantitatively compare the time costs of the two methods. we list the time cost increases of S_i_*Adaptation* and $MuNet$ separately when an extra stream is gradually added in Experiment c). According to the results in Table 3.3, the time cost is the same between S_i_*Adaptation* and $MuNet$ when there is only one stream S_0, as now $MuNet$ is exactly the same as S_i_*Adaptation*. Then we add the S_1 stream. The time cost of S_i_*Adaptation* increase from 87 to 167, and the time cost of $MuNet$ increase from 87 to 130. As a result, $MuNet$ is **37 minutes** faster S_i_*Adaptation*. Similarly, we add the S_2 stream. The time cost of S_i_*Adaptation* increase from 167 to 248, and the time cost of $MuNet$ increase from 130 to 176. $MuNet$ is **72 minutes** faster S_i_*Adaptation*. When the number of data streams increases to 10, it can be seen that the $MuNet$ already has a huge time advantage, which is almost **2.5 times** faster than S_i_*Adaptation*. Clearly, the time cost difference between S_i_*Adaptation* and $MuNet$ will be increasing greatly when

more streams are included, indicating our $MuNet$ is an efficient prediction method for multiple data streams with concurrent drift.

3.5 Summary

This chapter addresses the problem of concurrent concept drift in multiple streams with considering the correlations between data streams. An efficient method called $MuNet$ is designed to predict multiple streams. $MuNet$, one of the streams is randomly assigned as the base stream, predictors are retrained periodically for the base stream while the predicted value of other streams are corrected by an online learned Bayesian neural network called *BNN-connector*. As it takes less time to online learn *BNN-connector* than retraining predictor, $MuNet$ is able to predict multiple data streams with less computational cost as well as ensuring the prediction accuracy. We validated $MuNet$ on a real-world weather forecast dataset. The experimental results show that our method can efficiently save computational cost with high accuracy compared to traditional drift handling techniques for the single data stream.

CHAPTER 4

MULTI-STREAM CONCEPT DRIFT SELF-ADAPTATION USING GRAPH NEURAL NETWORK

4.1 Introduction

The explosion in the number of data sources generates fast-evolving data streams which have become one of the main data forms in current daily life [105]. Concept drift is a ubiquitous phenomenon in streaming scenarios. It refers to changes of data distribution in the data stream over time [106]. For example, a traffic accident causes distribution changes of traffic speed; topic changes cause distribution changes of the text in the media stream. Concept drift brings challenges to traditional machine learning algorithms as these algorithms assume the upcoming data has the same distribution as the historical data [107]. Such static learning algorithms will be invalid in this dynamically changing environment. When drift occurs, the learned model based on historical data cannot adapt to new data distribution and the prediction results will no longer be reliable[4].

Concept drift adaptation is to handle the concept drift problem and there are two main categories: informed and blind adaptation[53]. The informed adaptation, when to

update the predictor is determined by the feedback from a drift detection module[46]. For example, [108] use a sliding window to detect the drift and the predictor is updated when drift is detected. For blind methods, regardless of whether drift occurs, the model will be updated according to pre-designed rules, such as incremental learning-based adaptation methods [109, 110].

So far, most existing adaptation methods are designed for single data stream [111]. In practice, however, multi-stream is more common in most real scenarios [19]. If we treat each stream in multi-stream independently and simply apply the adaptation methods for single stream on them separately, the correlation between streams would be ignored [112]. As a result, the prediction results are less appropriate to support data-driven decision-making for multiple correlated streams.

Multi-stream environments have complex drift situations due to latent correlations between streams. With the consideration of possible concept drift issues in multi-stream, the correlation between data streams is more complex as well [113]. For example, when drift occurs on a data stream, other streams may soon or have faced a similar drift problem, namely, drifts may appear in multiple streams simultaneously or with a delay [79]. This is called a drift correlation between streams [111]. Constructing a drift adaptation framework for multiple correlated and drift-correlated streams has become a new challenge in the concept drift field. Several researchers have proposed some solutions [114]. For example, [115] proposes an adaptive framework for multi-stream classification, which sets the drift detection module on the source stream to update the classifier of the target stream. There is a similar stream setting in [116, 117]. The work in [111] focuses on the concurrent drift problem in multi-stream and proposes the MuNet framework which builds Bayes-based connectors between streams.

Current studies in multi-stream still have many deficiencies. First, some of them assume a source and target stream in their setting [116, 117]. However, in most cases,

there are no explicit source and target streams in real-world multi-stream. As a result, these methods are less applicable in large-scale multi-stream environments. Besides, current methods do not consider stream dependencies in-depth, or they only consider specific drift types.

To fill those gaps, we propose a self-adaptation learning framework for multi-stream using graph neural network (GNN), named SAGN. SAGN can handle different types of drift in multi-stream. It does not require the pre-knowledge of which streams are source or target. In SAGN, the dependencies between streams are presented in a graph structure by introducing GNN. GNN is well-known for deeply mining the correlations and dependencies among multiple nodes in the graph structure [118], but it is not originally designed to overcome the concept drift problem. To solve this issue, we reconsider the learning procedure of GNN-based methods from the aspect of concept drift adaptation. We convert the original prediction task into online streaming data tasks in sub-graphs. Each sub-graph handles concept drift that corresponds to an adaptation target locally. Globally, the entire graph presents the correlation between streams. Therefore, SAGN can solve the concept drift with considering the dependencies between streams.

To implement SAGN, we design an offline training stage and an online testing stage. The offline training stage assumes a fixed batch of historical data is available to train an initial model. In the online testing stage, new samples arrive one by one over time, and we update the initial model when every new sample arrives. The offline training uses historical data and Gumbel sampling to train an initial GNN. This GNN will be continuously updated (via sub-graphs) to adapt concept drift in the online testing stage. Instead of using one GNN, we choose the best GNN from a pool of GNNs. The GNN pool is a collection of multiple well-performed GNNs. The best GNN at each update is selected by testing with a dynamically updated validation set.

Our contributions are as follows:

- We propose a multi-stream prediction setting that multiple streaming data, containing multiple correlated streams, share the same time step. The correlation in multi-stream contains the dependencies between streams and the correlation between drifts. Compared to previous research, it does not need to assume the source/target stream.

- We design an online adaptation strategy for the GNN to handle different types of concept drift which is implemented locally in sub-graphs. An advantage of this design is that globally it maintains the correlation in multi-stream. The proposed dynamic validation set and GNN pool rolling mechanism are to work in tandem to achieve this process.

- We propose SAGN, a multi-stream concept drift self-adaptation framework using GNN. SAGN can overcome the concept drift problem in multi-stream with preserving the correlation between streams. Compared to existing methods, it is more applicable for large-scale multi-stream environments.

- We tested SAGN comprehensively on synthetic and real-world, drift or non-drift data. The experiment results provide a useful guide for future applications or future studies in this area.

The work described in this chapter has been published in the IEEE-TKDE paper "Multi-Stream Concept Drift Self-Adaptation Using Graph Neural Network".

4.2 Problem Statement

In this section, we first settle some important notations, definitions, and problem statements in Section 4.2.1. Also, we introduce the Gumbel-Softmax Trick method in Section 4.2.2 as the preliminaries that are used in our SAGN.

4.2.1 Problem Setting

In this section, the definitions and problem statement are introduced. We present a table of some important notations in Table 4.1 for brevity.

Table 4.1: Summary of Notations

Notations	Description
$\mathscr{S} = \{\mathscr{D}^1, \cdots, \mathscr{D}^s\}$	A multi-stream contains s data streams.
$\mathscr{D} = \{T^1, \cdots, T^g\}$	A data stream contains g time series.
$T^j = \{\mathrm{X}_t^j\}$	A time series in the data stream.
X^j	The feature value of the time series.
$t \in \mathbb{Z}^+$	The time step.
$\mathscr{P}$	The probability distribution.
p	The probability density function.
τ	A time period.
Δ	The past window size.
η	The prediction time steps.
$\mathscr{G}$	The correlation graph structure of multi-stream.
H	The predictor for multi-stream.
$\mathscr{S}^{tr}, \mathscr{S}^{val}, \mathscr{S}^{te}$	The training, validation and testing sets.
$P = \{H_1^i \mid i \in [C]\}$	The designed GNN pool with size C.
V^*	The designed dynamical validation set.
B_o	The size of V^*.

Definition 4.1 (Single Data Stream)**.** A data stream is formulated by $\mathscr{D} = \{T^1, T^2, \cdots, T^g\}$, and it contains g time series. For any $j \in [g]$, $T^j = \{\mathrm{X}_t^j | t = 1, \cdots, \infty\}$ is one time series in the stream $\mathscr{D}$, where X_t^j represents the feature value of the T^g in time step t. $T^j \in \mathscr{D}, j \in [g]$ is generated from distribution $\mathscr{P}_t$ with probability density function denoted as $p_t(\mathrm{X}_t^j)$.

Remark 4. *For example, consider a data stream generated by multiple sensors at a weather monitoring station. The time series produced by different sensors in this data stream share the same location information, and different time series represent different monitoring features.*

Definition 4.2 (Concept Drift in Single Data Stream [55])**.** Concept drift occurs in a data stream $\mathscr{D}$, if $\exists t_{g^{(i)}}$ that

$$\begin{cases} p_{t+1}(\mathscr{D}) \neq p_t(\mathscr{D}), \text{ for } t = t_{g^{(i)}} \\ p_{t+1}(\mathscr{D}) = p_t(\mathscr{D}), \text{ for } t \in \left(t_{g^{(i)}+\tau_i}, t_{g^{(i+1)}}\right) \end{cases}$$

where $\forall i,\ t_{g^{(i+1)}} - t_{g^{(i)}} > 1$, $t \in \mathbb{Z}^+$ represents the time step, $g^{(i)}$ represents the i^{th} drifted time point, and $1 < \tau_i < t_{g^{(i+1)}} - t_{g^{(i)}}$ is specifically for the occurrence of incremental drift.

Remark 5. *A data stream contains concept drift if the data distribution changes at least once, namely* $\left\{t_{g^{(i)}}\right\} \neq \emptyset$ *that* $p_{t+1}(\mathscr{D}) \neq p_t(\mathscr{D})$*, for* $t = t_{g^{(i)}}$*; in addition, the changed pattern is not ephemeral, but will last for a period (at least last for two time steps), which is manifested by* $\forall i,\ t_{g^{(i+1)}} - t_{g^{(i)}} > 1$*. The pattern stays the same in this period that* $p_{t+1}(\mathscr{D}) = p_t(\mathscr{D})$*, for* $t \in \left(t_{g^{(i)}+\tau_i}, t_{g^{(i+1)}}\right)$*; here* $\tau_i = 1$ *when the drift occurs suddenly while* $\tau_i > 1$ *when the drift occurs incrementally in the period of* $\left(t_{g^{(i)}+1}, t_{g^{(i)}+\tau_i}\right)$.

Suppose $T^{j'} \in \mathscr{D}$ is the target series, the prediction task for $\mathscr{D}$ refers to the prediction for $T^{j'}$. For example, there are many feature series in a weather station that presents a data stream, if the 'humidity' series is the target series, the prediction task for this stream refers to the prediction for the 'humidity' series.

Learning Aim at t-step for Single Data Stream: Let $\mathscr{D} = \{T^1, T^2, \cdots, T^g\}$ as a data stream, $T^{j'} = \{\mathrm{X}_t^{j'} | t = 1, \cdots, \infty\}$ as the target series. To obtain the prediction values of the next η time steps, the learning aim is to build a predictor H_t for $p_t(\mathscr{D})$ using the values in the past window of Δ steps, which can be denoted as

$$H_t = \underset{h \in \mathscr{H}}{\arg\min}\ \ell\left(h(\mathscr{D}_{t-\Delta+1:t}), \mathscr{D}_{t+1:t+\eta} \middle| \mathscr{D} \in p_t(\mathscr{D})\right),$$

where $\mathscr{H}$ is the hypothesis set, $\ell : \mathbb{R}^1 \times \mathbb{R}^1 \to \mathbb{R}_+$ is the loss function used to measure the magnitude of the error.

Definition 4.3 (Multi-Stream)**.** Multi-stream is a collection of multiple single data streams, and can be formulated by $\mathscr{S} = \{\mathscr{D}^1, \mathscr{D}^2, \cdots, \mathscr{D}^s\}$. Every stream $\mathscr{D}^s = \{T^1, T^2, \cdots, T^{g_s}\}$ in $\mathscr{S}$ contains several time series. Each T denotes one feature series, and there are g_s different features in total.

Remark 6. *Handling multi-stream problem is a big challenge. We propose to solve a multi-stream problem by each of the target features in the stream. Specifically, in this research, we assume that all streams in $\mathscr{S}$ contain the same features. This scenario is common in real applications. For example, many meteorological monitoring stations are set up in different city areas, where each station monitors multiple meteorological indicators (humidity, wind speed, temperature, etc.).*

Definition 4.4 (The Correlation in Multi-Stream)**.** The correlation in multi-stream includes the correlation between streams and a drift correlation that the occurrence of drift in one stream is related to the occurrence of drift in another stream. We use $\mathscr{G}_{true}$ to denote the correlation in multi-stream. For example, in a weather multi-stream, nearby weather stations share similar weather patterns. A sudden change of weather, for example, bushfire, will first lead to the drift of "humidity" in bushfire stations, and then other nearby stations will face a similar drift problem later.

Remark 7. *In most practical situations, it is hard for us to obtain $\mathscr{G}_{true}$. Because it is never known in advance when drift occurs and how severe it will be. In addition, $\mathscr{G}_{true}$ may also change over time rather than maintaining a fixed structure. Therefore, we need to estimate $\mathscr{G}_{true}$ during the learning procedure in multi-stream.*

Definition 4.5 (Concept Drift in Multi-Stream)**.** Let $\mathscr{S} = \{\mathscr{D}^1, \mathscr{D}^2, \cdots, \mathscr{D}^s\}$ is a multi-stream. A multi-stream $\mathscr{S}$ has concept drift if any data stream in $\mathscr{S}$ has concept drift.

Definition 4.6 (Concept Drift Correlation in Multi-stream)**.** Given $\mathscr{D}^u, \mathscr{D}^v \in \mathscr{S}$ (where $u \neq v$), p^u and p^v are the probability density function for stream $\mathscr{D}^u$ and $\mathscr{D}^v$, respectively. The drift correlation has two cases, concurrent drift and delayed drift.

A *concurrent drift* occurs if $\exists t_{g^{(i)}}$ that

$$\begin{cases} p^u_{t+1} \neq p^u_t, p^v_{t+1} \neq p^v_t \text{ for } t = t_{g^{(i)}} \\ p^u_{t+1} = p^u_t, p^v_{t+1} = p^v_t \text{ for } t \in \left(t_{g^{(i)}+\tau_i}, t_{g^{(i+1)}}\right) \end{cases}$$

A *delayed drift* occurs if $\exists t_{g^{(i)}}$, $\exists t_{g^{(j)}}$ and $i \neq j$ that

$$\begin{cases} p^u_{t_i+1} \neq p^u_{t_i}, p^v_{t_j+1} \neq p^v_{t_j}, \text{ for } t_i = t_{g^{(i)}}, t_j = t_{g^{(j)}} \\ p^u_{t_i+1} = p^u_{t_i}, p^v_{t_j+1} = p^v_{t_j}, \text{ for } t_i \in \left(t_{g^{(i)}+\tau_i}, t_{g^{(i+1)}}\right), \\ \qquad\qquad\qquad\qquad t_j \in \left(t_{g^{(j)}+\tau_j}, t_{g^{(j+1)}}\right) \end{cases}$$

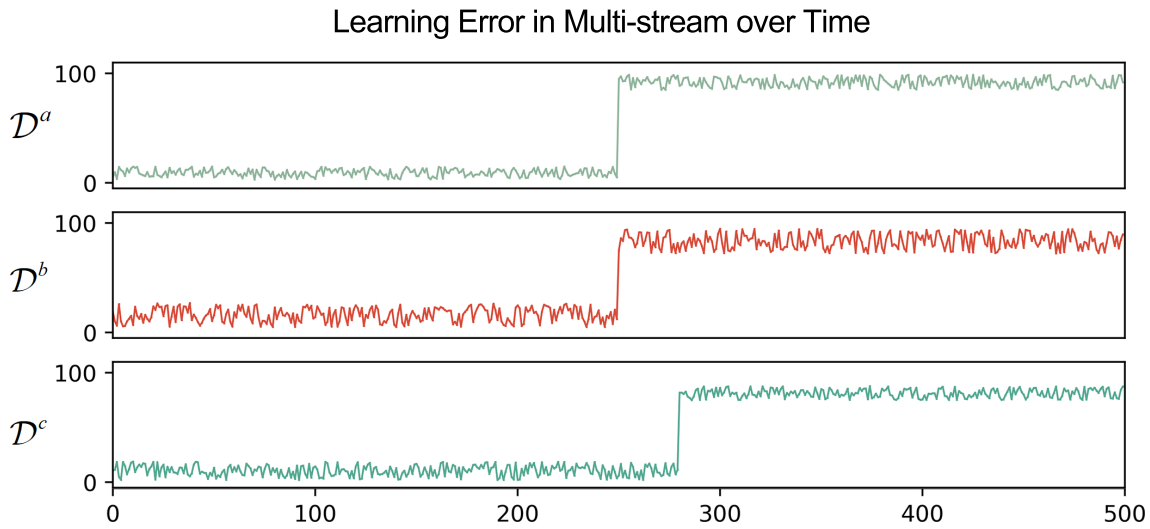


Figure 4.1: An example of concept drift correlation in multi-stream. Concurrent drift occurs in $\mathscr{D}^a$ and $\mathscr{D}^b$ ($t = 250$), and delayed drift occurs in $\mathscr{D}^c$ ($t = 280$).

Remark 8. *Concurrent drift means drifts occur in two or more streams in the multi-stream at the same time. If drifts occur with a time difference, this situation refers to delayed drift. We use the learning error of static predictors to explicitly represent two cases. When drift occurs, the error rates increase significantly due to the inability of the static*

predictors to adapt to the new data distribution. As shown in Fig. 4.1, Clearly, concurrent drift occurs in $\mathscr{D}^a$, $\mathscr{D}^b$ at $t = 250$ while delayed drift occurs in $\mathscr{D}^c$ at $t = 280$.

Remark 9. *It should also be noticed that the correlation in multi-stream $\mathscr{G}_{true}$ may change over time because of the occurrence of concept drift. For example, as shown in Fig. 4.2, the nodes $\mathscr{D}^1$, $\mathscr{D}^2$, $\mathscr{D}^3$, $\mathscr{D}^4$ and $\mathscr{D}^5$ represent five data streams in multi-stream separately and the edges in the graph structure represent correlations between two streams. It can be seen that the correlation remains stable globally, but if drift occurs in one of these streams at t step, such as $\mathscr{D}^4$, this drift may bring about changes in correlation locally, corresponding to changes in sub-graphs at $t+1$ step.*

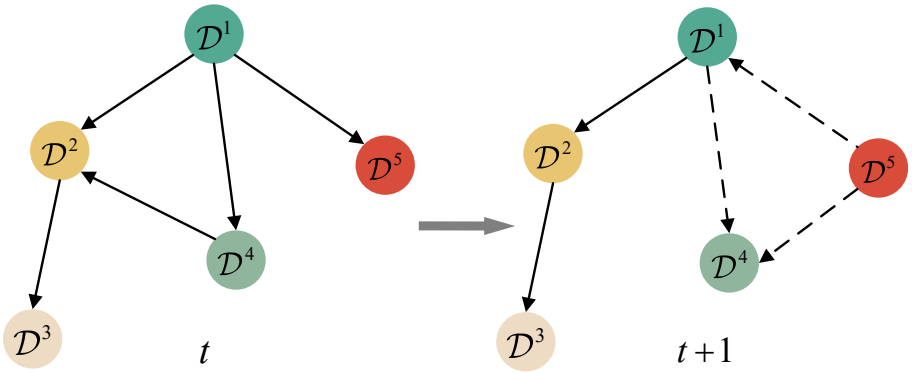


Figure 4.2: Correlation in multi-stream changes locally from t to $t+1$ due to the occurrence of concept drift.

Problem 1 (Learning for Multi-Stream at t-step with known $\mathscr{G}_{true}$)**.** *To obtain the prediction values of the next η time steps, the learning aim is to build a predictor H_t for $p_t(\mathscr{S})$ using the values in the past window of Δ steps. The learning process is based on the correlation $\mathscr{G}_{true}$ of the multi-stream $\mathscr{S}$, which can be denoted as*

$$H_t = \underset{h \in \mathscr{H}}{\arg\min}\ \ell\left(h(\mathscr{S}_{t-\Delta+1:t}, \mathscr{G}_{true}), \mathscr{S}_{t+1:t+\eta}\right)$$

Remark 10. *According to Problem 1, the learning objective in a multi-stream setting is to construct the predictor at each time point under the correlation in streams.*

If we use single-stream concept drift adaptation methods for the learning in multi-stream directly. Namely, we consider each stream separately and apply single-stream adaptation method one by one. The correlation $\mathcal{G}_{true}$ in multi-stream will be ignored. Besides, to obtain the prediction of s target features, we need to build s predictors for each separately. In this situation, Problem 1 is degenerated to

$$H_t^s = \underset{h^s \in \mathcal{H}^s}{\arg\min}\ \ell\left(h^s\left(D^s_{t-\Delta+1:t}\right), D^s_{t+1:t+\eta}\right).$$

Problem 1 represents an idealistic situation that $\mathcal{G}_{true}$ is known in advance. However, $\mathcal{G}_{true}$ is not available in most real cases. Therefore we propose to learn a correlation graph structure $\mathcal{G}_t$ to estimate $\mathcal{G}_{true}$. $\mathcal{G}_t$ is learned from the historical data to represent the correlation of the multi-stream $\mathcal{S}$. $\mathcal{G}_t$ will be continuously updated during the learning process. With considering the $\mathcal{G}_t$, the learning task has been changed, and the adaptation methods for single-stream are no longer applicable in this setting. To cover such a situation, we propose Problem 2.

Problem 2 (Learning for Multi-Stream at t-step with unknown $\mathcal{G}_{true}$)**.** *The learning aim for multi-stream at t-step with unknown $\mathcal{G}_{true}$ is to learn a correlation graph structure $\mathcal{G}_t$ and a predictor H_t for $p_t(\mathcal{S})$. The learning process is formalized as*

$$H_t, \mathcal{G}_t = \underset{h \in \mathcal{H}, \mathcal{G}}{\arg\min}\ \ell\left(h\left(\mathcal{S}_{t-\Delta+1:t}, \mathcal{G}_{t-1}\right), \mathcal{S}_{t+1:t+\eta}\right). \tag{4.1}$$

$\mathcal{G}_t$ is updated from $\mathcal{G}_{t-1}$ with every newly arrived sample in the online testing stage (Section 4.3.2). $\mathcal{G}_0$ is initialized on historical data using the Gumbel-Softmax trick (Section 4.2.2).

4.2.2 Gumbel-Softmax Trick

Although some methods [88, 95] are proposed to partially or completely avoid using the pre-defined graph, most of them still cannot achieve a high level of prediction accuracy

without any pre-defined graph information. To calculate the correlation graph structure of multi-stream without using the pre-defined graph completely, we use the Gumbel-Softmax trick [119, 120] to learn a sampled graph structure based on the historical data. This subsection will introduce this technique in detail.

In a discrete data space, whether a dimension is selected or not depends on whether the cumulative weight on that dimension is the maximum. In a multi-dimensional vector $\Omega = [\alpha_1, \alpha_2, \alpha_3, ...\alpha_i], i \in [N]$, whether a dimension is selected depends on whether the weight of that dimension is maximum. However, difficulties remain between converting the vectors into a probability-based representation and retaining computable gradients.

Normalizing a multi-dimensional vector by connecting a softmax function is a common way of dealing with this type of problem. By connecting the softmax function, the multi-dimensional vector can be normalized and the gradient calculation can be realized, but more probability information is lost. Sampling according to the probability distribution cannot perform the gradient calculation to update the network parameters. The Gumbel-Softmax technique is proposed to overcome the inability to apply the reparameterization technique to discrete data, and two results have been argued: 1) The discrete distributions can be parameterized by the Gumbel distribution; 2) The function corresponding to the Gumbel distribution can be continuous, benefiting from the continuous approximation property of the temperature parameter δ. When δ is reduced to zero, it can represent discontinuous expressions of the original distribution.

To make a discrete probability distribution meaningful rather than simply taking the value with the highest probability, and to be able to compute the gradient. The Gumbel trick is processed as follows:

First, perturbed Gumbel noises based on uniform distribution are generated which correspond to N dimensional vectors Ω. $\{\mathrm{G}um_i\}_{i \leqslant N}$, the i.i.d sequence of standard Gumbel random variables can be calculated by the following equation:

$$Gum_i = -\log(-\log(U_i)), U_i \sim U(0,1). \tag{4.2}$$

Then the discrete random variable Ω will be transformed as follows:

$$\Omega^{'} = [\log \alpha_1 + Gum_1, ... \log \alpha_i + Gum_i], i \in [N] \tag{4.3}$$

This means each dimension of the original vector adds the corresponding Gum_i to get the new vector $\Omega^{'}$. To scale discrete random variables to achieve continuity, a softmax map indexed by a temperature parameter is introduced, therefore the concrete distribution $\sigma_\delta(\Omega_i^{'})$ can be defined as:

$$\sigma_\delta(\Omega_i^{'}) = e^{\frac{\Omega_i^{'}}{\delta}} / \sum_{i=1}^{N} e^{\frac{\Omega_i^{'}}{\delta}} \tag{4.4}$$

where δ represents the temperature parameter, $\Omega_i^{'}$ represents the i-th dimensional vector of $\Omega^{'}$.

4.3 Methodology

In this section, our proposed method SAGN is introduced in detail. Fig. 4.3 provides an overview of the SAGN framework. The algorithm is divided into two stages: offline training and online testing. In the offline training stage, feature extraction is performed on the historical data and the Gumbel-Softmax trick is used to initialize the correlation graph structure $\mathscr{G}_0$ for multi-stream $\mathscr{S}$. After this, the $\mathscr{G}_0$ is embedded into the diffusion convolutional gated recurrent unit networks with a sequence-to-sequence structure. During the GNN training, multiple well-performed GNNs are stored to initialize the GNN pool. In the online testing stage, we design a dynamic online adaptation technique

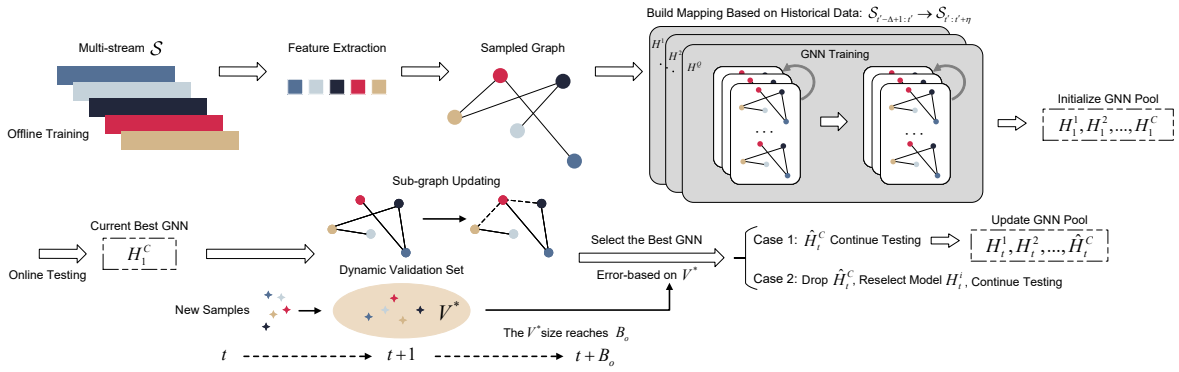


Figure 4.3: Overview of the SAGN framework. SAGN is divided into two stages. The offline stage extracts the feature and initializes a sampled graph structure that contains correlation in multi-stream $\mathcal{S}$. The online stage use arriving new samples to update sub-graphs to adapt to unknown concept drift. The dynamic GNN pool rolling mechanism is designed for reselecting the best GNN in a certain cycle, and GNNs stored are revalidated periodically on the proposed dynamic validation set over time.

to handle the concept drift problem. Specifically, the global correlation $\mathcal{G}_t$ of the multi-stream $\mathcal{S}$ is still preserved well and sub-graphs are updated by new samples to adapt to drift. Besides, we design a dynamically updated validation set and a GNN pool rolling mechanism to reselect the optimal GNN periodically to ensure stable prediction performance.

4.3.1 Initialization (Offline Training)

In this section, we will give the details of offline training. We first initialize a sampled correlation graph structure $\mathcal{G}_0$ using Gumbel-Softmax trick for multi-stream $\mathcal{S}$. After that, $\mathcal{G}_0$ will be embedded into GNN training and a GNN pool will be initialized.

4.3.1.1 Initialization of the Correlation Graph Structure

Most existing GNN-based methods use pre-defined graphs (usually containing the location information of sensors) of the data set as the basis for constructing the graph topology. Considering that pre-graphs of real-world data sets may not be available, we

construct the correlation graph structure based on historical data of multi-stream as the initialization.

First, We use the feature extraction network F_{etc} on multi-stream $\mathscr{S}$ to extract historical feature structure $F_{etc}(\mathscr{S})$ and reduce dimension. The feature extraction network F_{etc} consists of two convolutional layers, two normalization layers, and a fully connected layer [96].

After the calculation of F_{etc}, the data streams are combined in vector pairs, and the strength of the correlation between two streams is defined by probability $\mu_{\langle \mathscr{D}^u, \mathscr{D}^v \rangle} \in [0,1]$. The pairwise combination of vectors is obtained by concatenation and computation of two fully connected layers. The Gumbel-Soft trick is applied here to calculate the sampled graph structure on every two streams u and v: $G_{\langle \mathscr{D}^u, \mathscr{D}^v \rangle}$, which is based on the discrete probability variable $\mu_{\langle \mathscr{D}^u, \mathscr{D}^v \rangle}$ (see Eq. (4.2), (4.3) and (4.4)):

$$\mathscr{G}_{\langle \mathscr{D}^u, \mathscr{D}^v \rangle} = softmax\left(\frac{\log\left(\frac{\mu_{\langle \mathscr{D}^u, \mathscr{D}^v \rangle}}{1-\mu_{\langle \mathscr{D}^u, \mathscr{D}^v \rangle}}\right) + (\mathrm{G}um_u - \mathrm{G}um_v)}{\delta}\right) \tag{4.5}$$

Let $\mathscr{G}_0 = [\mathscr{G}_{uv}]_{s \times s}, u, v \in [s]$ denoted as initialized correlation graph structure, where $\mathscr{G}_{uv} = G_{\langle \mathscr{D}^u, \mathscr{D}^v \rangle}$ (corresponding to Step 3 in Algorithm 1). It can be seen as the initialization correlation graph structure of multi-stream $\mathscr{S}$ and would be embedded in the GNN training process.

4.3.1.2 Initialization of the GNN Pool

The sequence-to-sequence model [121], which has an encoder-decoder structure is applied to build the mapping relationship between $\mathscr{S}_{t-\Delta+1,t}$ and $\mathscr{S}_{t+1:t+\eta}$ during GNN training process. The initialization correlation graph structure $\mathscr{G}_0$ is embedded in the recurrent convolutional framework to handle all streams simultaneously.

The hidden state of the sequence-to-sequence model updates from $\Theta_{t^*-1} \rightarrow \Theta_{t^*}$ at each time step t^*. The values of $\mathscr{S}_{t^*}$ in the defined window size Δ are the input. Specifically,

the encoder part performs updates recurrently: $t^{'} - \Delta + 1 \rightarrow t^{'}$ and ends with Θ_{t^*} as the representation of $\mathscr{S}_{t^*}$. Correspondingly, the Θ_{t^*} is used for the initialization of the decoder. The hidden state performs the recurrent updates of each next η time steps in the future: $t^{'} + 1 \rightarrow t^{'} + \eta$.

We use the Diffusion Convolutional Gated Recurrent Unit defined in [86] and the concatenation method for the graph structure and hidden statements in [96] to collaboratively complete GNN training process (corresponding to Steps 4 to 7 in Algorithm 1):

$$\begin{cases} r_{t^*} = sigmoid(W_r \star_{\mathscr{G}_0} [\mathscr{S}_{t^*} \parallel \Theta_{t^*-1}] + f_r), \\ q_{t^*} = \tanh(W_q \star_{\mathscr{G}_0} [\mathscr{S}_{t^*} \parallel (r_{t^*} \odot \Theta_{t^*-1}] + f_q), \\ c_{t^*} = sigmoid(W_c \star_{\mathscr{G}_0} [\mathscr{S}_{t^*} \parallel \Theta_{t^*-1}] + f_c), \\ \Theta_{t^*} = c_{t^*} \odot \Theta_{t^*-1} + (1 - c_{t^*}) \odot q_{t^*} \end{cases}$$

where $\star\mathscr{G}_0$ represents the graph diffusion convolution and can be defined as:

$$A \star_{\mathscr{G}_0} W_\lambda = \sum_{k=0}^{K} (w_{k,1}^{\lambda} (M_O^{-1} \mathscr{G}_0)^k + w_{k,2}^{\lambda} (M_I^{-1} \mathscr{G}_0{}^T)^k) A$$

where r_{t^*} and q_{t^*} are the reset gate and update gate at time step t^*, and M_O and M_I are the out-degree and in-degree matrix. W_λ for $\lambda = r, q, c$ are the model parameters for the corresponding filters, and diffusion degree k is a hyperparameter. Mean absolute error (MAE) between the η prediction values and ground truth per Δ is used as the loss function of the training process.

The GNN training will iterate many times according to pre-setting, and the latest well-performed GNNs are stored to initialize the GNN pool: $P = \{H_1^i(\cdot; \mathscr{G}_0^i, \Theta_0^i) | i \in [C]\}$. $\mathscr{G}_0^i$ are updated from the GNN training process (corresponding to Step 8 in Algorithm 1). In this chapter, we store 20 GNNs in P. The scale of the GNN pool can be flexibly adjusted according to the actual hardware environment and the update speed of the multi-stream.

During the calculation of the graph convolution, both correlations and spatiotemporal dependencies of different streams are both taken into account. In addition, compared to only using the best GNN, the GNN pool can preserve richer historical data of multi-stream.

4.3.2 Dynamic Online Self-Adaptation (Online Testing)

In this section, we will give the details of online testing. The dynamically updated validation set and GNN pool rolling mechanism are proposed to work in tandem to achieve the dynamic self-adaptation process. Globally the correlation structure is maintained well and sub-graphs will be updated by new arriving samples locally to adapt to concept drift.

4.3.2.1 Dynamically Updated Validation Set

The GNNs stored in P represent the historical data information of multi-stream S. But these GNNs cannot adapt to concept drift in future time steps directly. So we design the dynamic self-adaptation in the online testing stage, the adaptation process is performed by updating sub-graphs using new samples. To maintain the prediction performance of the framework, we will reselect the Best GNN from P periodically.

Therefore revalidating the performance of GNNs is necessary. In SAGN, we design *a dynamically updated validation set* V^* with size B_o on the test set to achieve it. V^* is updated with new samples over time (corresponding to Step 9 in Algorithm 1). Specifically, Once the count of arriving new samples achieves B_o, the validation process will start. Multiple GNNs in P and the current updated GNN will be validated on the current V^* (corresponding to Step 16 in Algorithm 1). And the best GNN will be used to process the prediction task for coming samples in the next time steps. The setting of B_o depends on the data set characteristics. For example, if the data set has a slow change cycle, we can set it to a larger size. When it is used for a multi-stream that may change

rapidly, we appropriately reduce its size to ensure that the update cycle is maintained within the optimal learning period for the new data distribution.

The proposed dynamically updated validation set can ensure that the current GNN matches the latest data distribution, and the SAGN is able to maintain high-precision prediction performance for a long time.

4.3.2.2 Dynamic GNN Pool Rolling Mechanism

The updates for sub-graphs in SAGN are designed to adapt to concept drift, but sometimes erroneous updates are inevitable. To solve this problem, we design *a GNN pool rolling mechanism* to avoid this problem.

Specifically, suppose H_1^C is the best GNN in P, the online testing will start with H_1^C to predict the values of the next η time steps on the test set continuously. When the ground truth values of new samples are obtained, we use these values to update the sub-graphs of $\mathcal{G}_{t-1}^C$ to adapt to the unknown concept drift, and the GNN is also updated: $H_t^C \rightarrow \hat{H}_t^C, \mathcal{G}_{t-1}^C \rightarrow \mathcal{G}_t^C$ (corresponding to Step 11 to 14 in Algorithm 1). Therefore the GNN can learn and adapt to the drift. When the count of new sample arrival reaches B_o, the first dynamic validation set ${V_1}^*$ is generated automatically based on these new samples. Then, the loss values of all GNNs in P and $\hat{H}_t^C$ are validated on ${V_1}^*$. If $\hat{H}_t^C$ still reach the best performance on ${V_1}^*$, $\hat{H}_t^C$ replaces H_t^C, and P is updated. If not, for example, H_t^1 performs best on ${V_1}^*$, the current updated GNN $\hat{H}_t^C$ will be dropped. P would stay and H_t^1 is the prediction model in the next stage (corresponding to Step 16 in Algorithm 1). The rolling period of the mechanism depends on the size of B_o.

A stable prediction performance is guaranteed in this dynamic model selection and validation mechanism. Also, this enables SAGN to avoid performance degradation caused by erroneous updates.

The pseudo-code of SAGN is shown in Algorithm 1. Steps 1 to 8 are about the offline training stage. First, we set several pre-settings for initialization. Then, the training

process is performed after splitting the data set. Step 3 initializes the correlation graph structure $\mathcal{G}_0$. Steps 4 to 7 show the GNN iterative training process. The GNN pool is initialized with multiple well-performed GNNs as alternative models to the prediction task in the online self-adaptation stage in Step 8. Meanwhile, $\mathcal{G}_0$ is updated to $\mathcal{G}_t^i$ for these GNNs.

The dynamic self-adaptation process starts from Step 9. The prediction task is started from H_t^C and the updating for sub-graphs is performed simultaneously in Steps 10 to 14. When every dynamic validation set is generated, current GNNs in the pool will be revalidated and reselected again in Steps 15 to 17. This dynamic GNN pool rolling mechanism runs through the whole online testing stage. The overview of SAGN framework can be seen in the Fig. 4.3.

4.4 Experiments

In this section, we performed extensive experiments to demonstrate that the proposed SAGN outperforms existing prediction frameworks, including traditional methods, the adaptation method, and GNN-based methods in most cases. Furthermore, we construct synthetic data sets to demonstrate that SAGN can learn and adapt when severe drift occurs in multi-stream.

We first give details of three real-world data sets. The experiment settings, comparative baselines, and experiment results are also given in the corresponding subsection.

4.4.1 Data Sets

Three real-world large-scale benchmark data sets are used for our experiments, and the summary information of data sets is shown in Table 4.2.

(1)**METR-LA**: This is a traffic information data set that was collected from the loop detection sensors in highways in Los Angeles Country[123]. Four months of speed data

Algorithm 1 SAGN

1: Input Multi-stream $\mathscr{S}$, the window size of historical sequence for all streams Δ, the prediction time steps for coming samples η, training learning rate LR and epoch Q_{max}, online self-adaptation learning rate lr and epoch $Q^{'}_{max}$, the temperature for Gumbel sampling δ, loss function (MAE is selected) and optimization algorithm (Adam algorithm [122] is selected);

2: Split we divide the whole multi-stream data set into training $\mathscr{S}^{tr}$, validation $\mathscr{S}^{val}$ and test $\mathscr{S}^{te}$ sets along the timeline, B_t is the batch size for $\mathscr{S}^{tr}$, B_o is the size of the dynamic validation set V^*;

3: Initial Θ_0, $\mathscr{G}_0$ (see Eq.(4.5), $P = \emptyset$

for $Q = 1,2,\ldots,Q_{max}$ **do** // Offline training based on historical data

4: Fetch training data from $\mathscr{S}^{tr}$ with batch size B_t;

5: Calculate $L = loss(H^Q\left(\mathscr{S}^{tr}_{t'-\Delta+1:t'};\mathscr{G}_0,\Theta_0\right),\mathscr{S}^{tr}_{t'+1:t'+\eta})$;

6: Update $\mathscr{G}_0,\Theta_0 = \text{Adam}(L)$;

7: Calculate the loss on $\mathscr{S}^{val}$: $L_Q = loss(H^Q\left(\mathscr{S}^{val}_{t'-\Delta+1:t'};\mathscr{G}_0,\Theta_0\right),S^{val}_{t'+1:t'+\eta})$ and save the model H^Q;

end

8: Initialize GNN Pool

select C well-performed GNNs from $\{H^1,H^2,\cdots,H^{Q_{max}}\}$ and initialize the GNN pool: $P = \{H^i_1(\cdot;\mathscr{G}^i_0,\Theta^i_0)|i \in [C]\}$. Suppose H^C_1 is the best one;

for *new samples in* $\mathscr{S}_{te}$ *arrive in each time step,* $t = 1,2,3,\ldots$ **do** // Online testing and adaptation

9: Generate Dynamic Validation Set new samples are stored in V^* temporarily and its size up to B_o;

10: Online Prediction $H^C_t\left(\mathscr{S}^{te}_{t-\Delta+1:t};\mathscr{G}^C_{t-1},\Theta^C_{t-1}\right) = S^{pre}_{t+1:t+\eta}$; // Give online prediction results

if $t > 1$ **then**

11: Obtain ground truth: $S^{true}_{t:t+\eta-1}$;

for $Q^{'} = 1,2,\ldots Q^{'}_{max}$ **do** // Sub-graphs updating

12: Calculate $L = loss(\hat{H}^C_t\left(S^{te}_{t-\Delta:t-1};\mathscr{G}^C_{t-1},\Theta^C_{t-1}\right),S^{true}_{t:t+\eta-1})$;

13: Update $\mathscr{G}^C_{t-1},\Theta^C_{t-1} = \text{Adam}(L)$, H^C_t;

end

end

14: Update $\mathscr{G}^C_t = \mathscr{G}^C_{t-1}$, $\hat{H}^C_t = H^C_t$;

end

if *the size of* V^* *reaches* B_o **then**

15: Calculate $L_i = loss(H^i_t\left(V^*_{t-\Delta+1:t};\mathscr{G}^i_{t-1},\Theta^i_{t-1}\right),V^*_{t+1:t+\eta})$, $i \in [C]$; $\hat{L}_C = loss(\hat{H}^C_t\left(V^*_{t-\Delta+1:t};\mathscr{G}^C_{t-1},\Theta^C_{t-1}\right),V^*_{t+1:t+\eta})$

16: Validate and reselect the GNN

if $\hat{L_C} < \arg\min_{i\in[C]}\{L_1,L_2,..L_C\}$ **then** // Dynamic model selection

$\hat{H}^C_t$ is selected for the prediction GNN in the next stage;

$H^C_t = \hat{H}^C_t$, update the GNN pool P;

else

drop: $\hat{H}^C_t$;

select he model H^i_t corresponding to $\arg\min_{i\in[C]}\{L_1,L_2,..L_C\}$ as the prediction GNN for the next stage;

end

17: reset V^*

end

Table 4.2: Summary Statistics of Data sets

Data set	Nodes	Samples	Sample Rate
METR-LA	207	34272	5min
PEMS-BAY	325	52116	5min
WEATHER ($psur$)	10	29639	1h
WEATHER ($q2m$)	10	29639	1h

streams (Mar 31_{st} 2012 - Jun 30_{st} 2012) generated by 207 sensors.

(2)**PEMS-BAY**: This is a traffic data set collected by the California Transportation Agencies (CalTrans) Performance Measurement System [124]. Six months of speed data streams (Jan 1_{st} 2017 - May 31_{st} 2017) generated by 325 sensors.

(3)**WEATHER FORECASTING**: This is a publicly available dataset compiled from weather stations in Beijing, China [125]. In the remainder of this chapter, it is abbreviated as WEATHER. This data set comprises hourly weather data from ten weather stations (from Jan 1_{st} 2015 - May 31_{st} 2018, which is 1188 days in total). It comprises weather monitoring features from 3 : 00 intraday to 15 : 00 the next day. Each weather station has the same weather monitoring features. We selected two weather features $psur$ and $q2m$ for the experiments. They represent the surface air pressure (unit ,$\tilde{N}\acute{E}$) and the specific humidity at a height of two meters above the ground (unit g/kg), respectively.

The same data pre-processing procedure as in [86] is performed for METR-LA and PEMS-BAY data sets. For WEATHER, the data preprocessing is performed by removing the overlapping parts on the timeline, and the missing values are replaced by the mean of the preceding and following time steps. We apply Z-Score normalization to all inputs. For METR-LA and PEMS-BAY, 70% of the data is used for training, 10% is used for validation, and the remaining 20% is used for testing. For WEATHER, 40% of the data is used for training, 10% is used for validation, and the remaining 50% is used for testing. The training and validation batch size is 64 for all data sets, and we simulate the online multi-stream environment in the form of samples arriving sequentially at a single time

step.

4.4.2 Experiment Settings

4.4.2.1 Baseline Methods

Since the current research on multi-stream concept drift is still not well solved in the field, the aim of this chapter is to develop a solution to handle it. Some existing GNN-based methods are chosen for comparison because our study reconsiders the learning procedure of GNN-based predictors from an aspect of concept drift adaptation for multi-stream. In addition, some traditional models based on statistics and the adaptation method designed for multi-stream are considered accordingly:

- Traditional non-deep learning methods:

  (1)**HA**: Historical Average, which achieves the prediction task using the weighted average of historical data;

  (2)**ARIMA**$_{kal}$: Auto-Regressive Integrated Moving Average model with the Kalman filter;

  (3)**VAR**: Vector Auto-Regression;

  (4)**SVR**: Support Vector Regression, which uses a linear support vector to achieve the regression.

- Traditional deep learning methods:

  (1)**FNN**: Feedforward Neural Network, which includes two hidden layers and $L2$ regularization;

  (2)**FC-LSTM**: Fully Connected LSTM, which means the recurrent neural network is connected to the LSTM hidden units.

- Multi-stream adaptation method:

  (1)**MuNet** [111]: A Bayesian neural network-based multi-stream adaptation frame-

work. Due to the limitation of the connector on the expansion performance, the algorithm can only be effective for the concurrent drift problem in multi-stream with a small data scale. For the fairness of comparison, we put SAGN and MuNet in the same experiment setting to compare. Specifically, we choose the same ten streams from METR-LA and PEMS-BAY data sets separately for these two adaptation methods (we use SAGN (10) and MuNet (10) to represent them).

- GNN-based methods:

  (1)**Graph-WaveNet** [88]: Graph WaveNet learns an adaptive dependency matrix through node embedding, combining graph convolution and spatiotemporal convolution to capture the temporal and spatial dependencies ;

  (2)**DGCRN** [91]: Dynamic Graph Convolutional Recurrent Network integrates dynamic graphs generated by dynamic filters at each time step and pre-defined graphs into a dynamic convolutional recurrent network to perform the prediction task;

  (3)**ASTGCN** [90]: Attention-based Spatial-Temporal Graph Convolution Networks design the spatial-temporal attention mechanism to synergistic graph convolution for weighted fusion of temporal and spatial features;

  (4)**AGCRN** [95]: Adaptive Graph Convolutional Recurrent Network designs two adaptive blocks of node adaptive learning and data adaptive graph to automatically infer the dependencies between streams;

  (5)**DCRNN** [86]: Diffusion Convolutional Recurrent Neural Network are designed for performing the graph convolution;

  (6)**GTS** [96]: Graph for Time Series designs the discrete graph learning structure to capture the dependencies between streams.

Table 4.3: The performance comparison between SAGN and the baseline methods on real-world data sets

METR-LA	Horizon 3			Horizon 6			Horizon 12		
Methods | Metrics	MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE
HA	4.16	7.80	13.00%	4.16	7.80	13.00%	4.16	7.80	13.00%
ARIMA	3.99	8.21	9.60%	5.15	10.45	12.70%	6.90	13.23	17.40%
VAR	4.42	7.89	10.20%	5.41	9.13	12.70%	6.52	10.11	15.80%
SVR	3.99	8.45	9.30%	5.05	10.87	12.10%	6.72	13.76	16.70%
FNN	3.99	7.94	9.90%	4.23	8.17	12.90%	4.49	8.69	14.00%
FC-LSTM	3.44	6.30	9.60%	3.77	7.23	10.90%	4.37	8.69	13.20%
Graph-WaveNet	2.69	5.15	6.90%	3.07	6.22	8.37%	3.53	7.37	10.10%
DGCRN	2.62	5.01	6.63%	2.99	6.05	8.02%	3.44	7.19	9.73%
ASTGCN	4.86	9.27	9.21%	5.43	10.61	10.13%	6.51	12.52	11.64%
AGCRN	2.87	5.58	7.70%	3.23	6.58	9.00%	3.62	7.51	10.38%
DCRNN	2.77	5.38	7.30%	3.15	6.45	8.80%	3.60	7.59	10.50%
GTS	2.64	4.95	6.80%	3.01	5.85	8.20%	3.41	6.74	9.90%
MuNet (10)	10.02	10.02	15.73%	10.06	10.06	15.79%	10.13	10.13	15.97%
SAGN (10)	2.56	3.98	6.60%	2.95	4.71	7.92%	3.50	5.67	9.69%
SAGN	**2.46**	**4.41**	**6.41%**	**2.79**	**5.15**	**7.64%**	**3.16**	**5.91**	**9.10%**

PEMS-BAY	Horizon 3			Horizon 6			Horizon 12		
Methods | Metrics	MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE
HA	2.88	5.59	6.80%	2.88	5.59	6.80%	2.88	5.59	6.80%
ARIMA	1.62	3.30	3.50%	2.33	4.76	5.40%	3.38	6.50	8.30%
VAR	1.74	3.16	3.60%	2.32	4.25	5.00%	2.93	5.44	6.50%
SVR	1.85	3.59	3.80%	2.48	5.18	5.50%	3.28	7.08	8.00%
FNN	2.20	4.42	5.19%	2.30	4.63	5.43%	2.46	4.98	5.89%
FC-LSTM	2.05	4.19	4.80%	2.20	4.55	5.20%	2.37	4.96	5.70%
Graph-WaveNet	1.30	2.74	**2.73%**	1.63	3.70	3.67%	1.95	4.52	4.63%
DGCRN	**1.28**	2.69	2.66%	**1.59**	3.63	**3.55%**	1.89	4.42	4.43%
ASTGCN	1.52	3.13	3.22%	2.01	4.27	4.48%	2.61	5.42	6.00%
AGCRN	1.37	2.87	2.94%	1.69	3.85	3.87%	1.96	4.54	4.64%
DCRNN	1.38	2.95	2.90%	1.74	3.97	3.90%	2.07	4.74	4.90%
GTS	1.32	2.62	2.80%	1.64	3.41	3.60%	1.91	3.97	4.40%
MuNet (10)	5.26	5.26	18.36%	5.31	5.31	18.46%	5.46	5.46	18.71%
SAGN (10)	1.58	2.43	3.77%	2.08	3.30	5.36%	2.55	4.06	6.93%
SAGN	**1.30**	**2.43**	**2.75%**	**1.60**	**3.11**	**3.61%**	**1.85**	**3.61**	**4.37%**

WEATHER ($psur$)	Horizon 3			Horizon 6			Horizon 12		
Methods | Metrics	MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE
HA	28.52	39.96	3.00%	28.52	39.96	3.00%	28.52	39.96	3.00%
ARIMA	2.16	2.16	0.22%	2.91	2.91	0.29%	4.30	4.30	0.43%
VAR	1.56	4.60	0.16%	2.30	5.36	0.23%	3.09	6.02	0.31%
SVR	1.62	5.21	0.16%	2.57	6.29	0.26%	3.58	7.21	0.36%
FNN	5.16	7.27	0.53%	5.94	8.10	0.60%	6.98	9.15	0.70%
FC-LSTM	2.06	5.00	0.21%	2.86	5.80	0.29%	3.71	6.54	0.38%
Graph-WaveNet	1.00	4.80	**0.10%**	1.44	5.13	**0.15%**	2.25	5.57	**0.23%**
DGCRN	**0.94**	4.33	**0.10%**	1.56	5.09	0.16%	2.71	6.11	0.28%
ASTGCN	1.52	4.50	0.14%	2.15	5.32	0.22%	2.90	5.94	0.30%
AGCRN	1.03	4.77	0.11%	1.48	5.11	**0.15%**	2.31	5.60	0.24%
DCRNN	1.06	4.81	0.11%	1.51	5.23	0.16%	2.37	5.24	0.24%
GTS	1.04	2.86	0.11%	1.55	3.39	0.16%	2.49	4.30	0.25%
MuNet	3.42	3.42	0.35%	3.45	3.45	0.35%	3.58	3.58	0.36%
SAGN	**0.96**	**1.29**	**0.10%**	**1.43**	**1.79**	**0.15%**	**2.29**	**2.67**	**0.23%**

WEATHER ($q2m$)	Horizon 3			Horizon 6			Horizon 12		
Methods | Metrics	MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE
HA	5.15	6.29	191.12%	5.15	6.29	191.12%	5.15	6.29	191.12%
ARIMA	**0.13**	**0.13**	12.66%	**0.20**	**0.20**	20.34%	**0.25**	**0.25**	24.09%
VAR	0.61	1.04	12.64%	0.86	1.34	18.84%	1.17	1.72	27.73%
SVR	0.25	0.58	13.32%	0.38	0.74	21.32%	0.53	0.89	32.70%
FNN	0.85	1.01	65.14%	1.00	1.18	79.11%	1.19	1.37	95.90%
FC-LSTM	0.30	0.58	19.28%	0.43	0.73	29.07%	0.59	0.88	42.29%
Graph-WaveNet	0.56	1.00	**10.38%**	0.79	1.30	**15.30%**	1.10	1.70	**23.33%**
DGCRN	0.56	1.01	10.89%	0.81	1.34	16.61%	1.17	1.79	25.83%
ASTGCN	0.61	1.02	13.39%	0.86	1.34	17.63%	1.16	1.72	25.11%
AGCRN	0.59	1.03	11.20%	0.82	1.33	16.13%	1.16	1.74	25.00%
DCRNN	0.59	1.03	11.20%	0.82	1.34	16.45%	1.16	1.77	24.98%
GTS	0.60	0.88	11.26%	0.84	1.17	16.47%	1.16	1.53	24.43%
MuNet	1.33	1.33	42.04%	1.35	1.35	42.55%	1.55	1.55	46.65%
SAGN	**0.56**	**0.69**	**10.72%**	**0.79**	**0.93**	**16.02%**	**1.10**	**1.25**	**24.24%**

4.4.2.2 Settings

Platform. We use PyTorch 1.7.1 to implement the model. All experiments are run on a server with the Nvidia Quadro RTX 8000 GPU (48GB). The CPU is Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz, and the total available memory of the server is 187GB.

Parameter Settings. The temperature parameter δ of the Gumbel-Softmax trick is set to 0.9. The Adam optimizer is selected to train the model. The initial learning rate is set to 0.005 for METR-LA, WEATHER and all synthetic data sets, and 0.001 for PEMS-BAY. The training epoch is set to 200. The initial learning rate was chosen according to the complexity of each dataset, and an early stopping strategy was employed for iterative training across all datasets. The online learning rate e for dynamic online self-adaptation is set to 0.00001 for METR-LA, and 0.000005 for PEMS-BAY and WEATHER. The size of the dynamic validation set (B_o) is set to 144 for METR-LA, PEMS-BAY, and 24 for WEATHER. The online learning epoch number is set to 3 for all data sets. Since the dataset WEATHER does not have a pre-defined graph, therefore when comparing the methods using pre-defined graphs, we use the nearest neighbor matrix calculated by $neighbors\_graph$ in Python package $sklearn$ instead of pre-defined graphs. The k is set to 5. The size of the GNN pool is set to 20 for all data sets.

Evaluation Metrics. The predicted values are transformed back to the actual values first and we compare them with the ground truth. Mean absolute error (MAE), mean percentage absolute error (MAPE) and root mean square error (RMSE) are used for the evaluation metrics for all experiments. To ensure the fairness of the comparison, the results on all data sets are the average of five repeated times.

4.4.3 Result Analysis

4.4.3.1 Overall Performance

Table 4.3 compares the performance of SAGN against all baseline methods on three real-world data sets. SAGN achieves state-of-art prediction performance in most cases. On the more commonly used traffic prediction datasets, the improvement is about 7% on METR-LA, and about 3% on PEMS-BAY. Especially in multi-step prediction, SAGN has obvious advantages. Since the vehicle speed recorded by the two traffic data sets shows a periodic change from day to night, learning historical data is useful for the prediction task. However, unexpected accidents on certain roads will cause drift to varying degrees and ranges, and this drift information will not be able to be learned from historical data. We also give the prediction results of the two weather indicators ($pusr$ and $q2m$) in the WEATHER data set. Compared to the above two traffic data sets, the weather data presents a more regular change trend and fluctuation range. This is the reason why some traditional regression prediction methods are still applicable. But despite this, SAGN's performance on the prediction tasks on the two weather indicators still exceeds most methods. For comparison with the multi-steam adaptation method MuNet, SAGN can also achieve great performance even on a small data scale.

4.4.3.2 Ablation Study

We construct synthetic data sets to demonstrate the self-adaptation effectiveness of SAGN, especially for sudden drifts that diverge significantly from the original distribution [99]. We design two fundamental multi-stream data sets, $\mathscr{S}^a$ and $\mathscr{S}^b$. In $\mathscr{S}^a$, there is no correlation between streams, and there is a correlation between streams in $\mathscr{S}^b$. Based on this setting, we add different types of functions to simulate drift at a certain time step $\hat{t}$ on the two multi-streams, and the drift-inducing functions cover multiple function types.

The first-class multi-stream $\mathscr{S}^a$ is initialized by the function $RegressionGenerator$ in the Python package $scikit-multiflow$. There are ten streams in $\mathscr{S}^a$ and the sample number of all streams is 30,000. Ten streams have different parameter settings of $ramdom\_state$ (from 0 to 9). To further distinguish the data streams, we add different degrees of offsets to all streams on this basis, with offsets of $20, 40, 70, 60, 15, 45, 57, 82, 26, 33$. These offsets are set randomly and have no special meaning. The second-class multi-stream $\mathscr{S}^b$ is derived from the real-world data set PEMS-BAY. Specifically, the data streams in PEMS-BAY are related to each other, so we transform on the part of PEMS-BAY to construct $\mathscr{S}^b$. We select the first 30,000 samples of ten streams in PEMS-BAY as the initialization for $\mathscr{S}^b$.

The sudden drift starts at the time step of the 15001_{st} sample in all streams of both $\mathscr{S}^a$ and $\mathscr{S}^b$. Multiple drift function types are considered to ensure completeness of the verification process, including linear, trigonometric, and polynomial functions. The details of these functions are defined as follows:

$$\begin{cases} \mathscr{S}^*_{lir} = 5\mathscr{S}^* + 50 \\ \mathscr{S}^*_{tri} = \sin(5\mathscr{S}^*) + 50 \\ \mathscr{S}^*_{pol} = 0.01(\mathscr{S}^*)^{2.5} - 0.1(\mathscr{S}^*)^2 + 0.5\mathscr{S}^* \end{cases} \tag{4.6}$$

Each time step is separated by one hour for all synthetic data sets. Therefore, we generate a total of eight synthetic multi-stream data sets according to the above settings. They are $\mathscr{S}^a_{ndf}$, $\mathscr{S}^a_{lir}$, $\mathscr{S}^a_{tri}$, $\mathscr{S}^a_{pol}$, and $\mathscr{S}^b_{ndf}$, $\mathscr{S}^b_{lir}$, $\mathscr{S}^b_{tri}$, $\mathscr{S}^b_{pol}$. The subscript ndf represents that no drift function is added in the multi-stream. lir, tri, and pol represent the different drift functions (as shown in Eq. (4.6)).

The ablation study experiment settings are as follows. For all synthetic data sets, We use 40% for training, 10% for validation, and the last 50% for testing. The learning rate of the dynamic online self-adaptation is 0.005 for $\mathscr{S}^a_{lir}$, 0.0005 for $\mathscr{S}^a_{tri}$, $\mathscr{S}^a_{pol}$, $\mathscr{S}^b_{lir}$, $\mathscr{S}^b_{pol}$,

0.00005 for $\mathscr{S}^b_{ndf}$, $\mathscr{S}^b_{tri}$, 0.000005 for $\mathscr{S}^a_{ndf}$. The size of the dynamic validation set is 144 for $\mathscr{S}^b_{ndf}$ and 24 for the others.

The experiment results are shown in Table 4.4. We select the GTS method as the baseline to compare in this part because it is the best-performance GNN model without using any pre-defined graph information. It can be seen that the GTS maintains good prediction performance when the multi-stream does not drift. But when the data distribution changes drastically after a certain time step, it is difficult for the GTS to adapt. Judging from the prediction results, GTS suffers severe performance slippage in multi-stream with sudden drift, while SAGN is not affected very much. Precisely, after building a correlation graph structure globally, SAGN maintains good prediction performance by continuously updating sub-graphs with arriving new samples. SAGN is able to learn the new data distribution after concept drift through a dynamic online adaptation process.

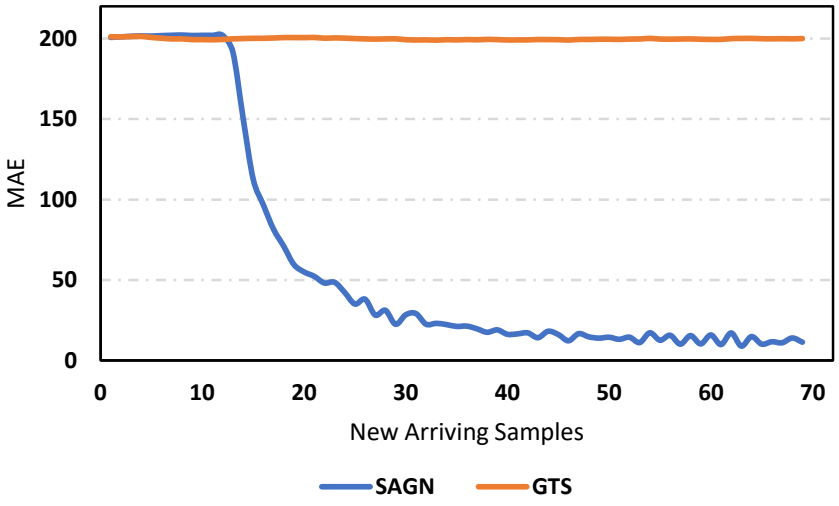


Figure 4.4: The adaptation performance comparison

In practical situations, the adaptation to sudden drift is particularly important, especially for the pre-judgment of some important detection indicators. Rapid adaptation means that it can quickly help decision-making and avoid major losses. To show the

adaptation performance of the proposed SAGN more concretely, we draw the figure of the MAE change when concept drift occurs on multi-stream. The comparison experiment is tested on the synthetic data set $\mathscr{S}^a_{lir}$. The results are shown in Fig. 4.4, SAGN can quickly adapt to the new data distribution after about ten new samples arrive, while GTS cannot.

Table 4.4: The performance comparison for SAGN and GTS under sudden drift

		Horizon 3			Horizon 6			Horizon 12		
Data Sets	Methods	MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE
$\mathscr{S}^a_{ndf}$	GTS	0.80	1.00	**2.34%**	0.80	1.00	**2.34%**	0.80	1.00	**2.34%**
	SAGN	**0.80**	**0.97**	2.35%	**0.80**	**0.97**	2.35%	**0.80**	**0.97**	2.35%
$\mathscr{S}^a_{lir}$	GTS	199.86	218.02	81.97%	199.23	217.35	81.70%	199.31	217.11	81.92%
	SAGN	**5.40**	**6.65**	**2.73%**	**5.40**	**6.64**	**2.73%**	**5.43**	**6.69**	**2.75%**
$\mathscr{S}^a_{tri}$	GTS	5.47	6.98	10.92%	5.81	7.72	11.62%	8.90	10.98	17.40%
	SAGN	**0.67**	**0.76**	**1.34%**	**0.67**	**0.76**	**1.34%**	**0.67**	**0.76**	**1.35%**
$\mathscr{S}^a_{pol}$	GTS	27.70	28.09	155.04%	27.44	27.91	153.48%	27.23	27.73	152.37%
	SAGN	**0.62**	**0.74**	**3.47%**	**0.62**	**0.73**	**3.46%**	**0.62**	**0.74**	**3.49%**
$\mathscr{S}^b_{ndf}$	GTS	9.00	16.56	3.74%	12.79	24.05	5.71%	18.06	32.94	8.59%
	SAGN	**8.67**	**13.20**	**3.61%**	**12.11**	**18.83**	**5.43%**	**16.50**	**25.41**	**8.02%**
$\mathscr{S}^b_{lir}$	GTS	53.76	57.01	43.07%	50.88	56.07	42.65%	52.47	59.91	44.79%
	SAGN	**7.67**	**10.06**	**7.27%**	**9.11**	**12.48**	**9.02%**	**11.30**	**15.43**	**11.40%**
$\mathscr{S}^b_{tri}$	GTS	96.20	96.38	192.45%	144.09	144.26	288.22%	135.84	136.65	271.68%
	SAGN	**0.95**	**1.10**	**1.78%**	**0.98**	**1.14**	**1.84%**	**1.01**	**1.16**	**1.86%**
$\mathscr{S}^b_{pol}$	GTS	164.91	170.57	517.10%	215.50	222.99	668.66%	243.28	249.56	747.18%
	SAGN	**3.19**	**4.52**	**28.34%**	**3.67**	**5.37**	**39.07%**	**4.36**	**6.44**	**54.13%**

We also tested the impact of the temperature parameter δ in Gumbel-Softmax trick. We report the performance on the METR-LA data set. This parameter determines the smoothness of sampling results. The smaller this parameter, the more sampling results

will tend to one-hot encoding, but the gradient changes greatly. The higher this parameter, the smoother the sampling result, but the variance of the gradient is small [119]. Fig. 4.5 shows the effect of this parameter.

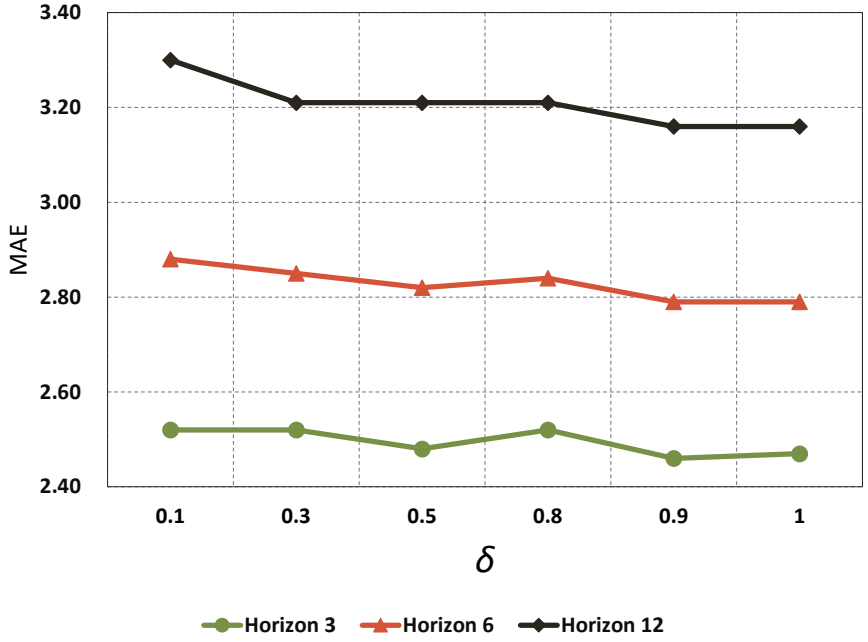


Figure 4.5: The effect of the δ in Gumbel-Softmax trick

Table 4.5: The Runtime Comparison

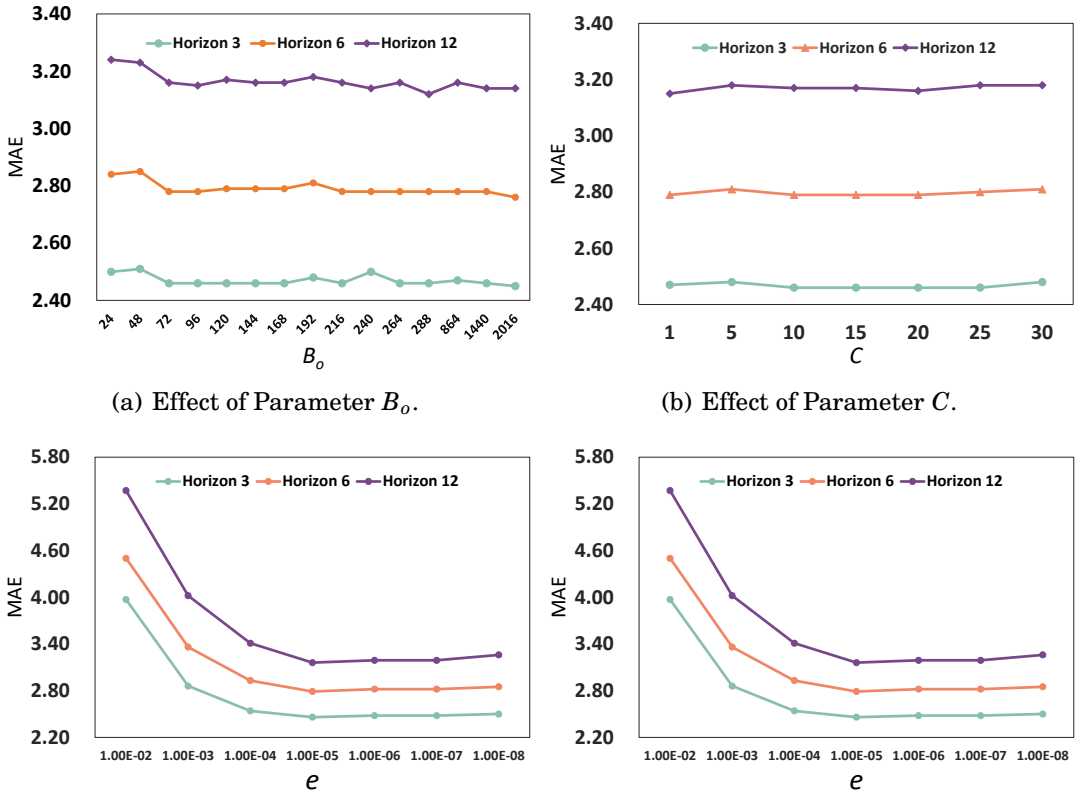
Method	METR-LA	
	training time(s/epoch)	testing time(s/samples)
SAGN	48.87	0.34
GTS	49.24	0.0006

The main difference between SAGN and other GNN-based methods is that our testing stage runs online. The concept drift self-adaptation progress is achieved by sub-graphs updating when new samples arrive. We compare the run time (as shown in Table 4.5) with GTS in the training and testing stages separately. The experiment is performed on the METR-LA data set. In the training stage, our average training time per epoch is slightly

shorter than GTS. In the test stage, neither GTS nor other traditional GNN-based methods consider concept drift adaptation, and the prediction modes are all performed on batch data in offline mode. But SAGN predicts the incoming new samples one by one according to the data stream online mode, and the average prediction time is $0.34s$, which is also much smaller than a single time step.

4.4.3.3 Parameter Sensitivity

We tested the parameter sensitivity of SAGN. We report MAE on the METR-LA data set. Whenever we change a test parameter, the other parameter values will keep their original default values.



(a) Effect of Parameter B_o.

(b) Effect of Parameter C.

(c) Effect of Parameter e.

(d) Effect of online learning epoch number.

Figure 4.6: The effect of different parameters in the adaptation stage.

First, we tested parameter B_o, which is the size of the dynamic validation set. This

size means the period of revalidation and reselection of the model. Generally speaking, for data streams that may change drastically, we can reduce this value to shorten the period of model revalidation. For data streams that may have periodic changes or a slow overall rate of change, we can appropriately increase this value to save computing resources. The dynamic validation set ensures that SAGN neither misses learning of new sample distributions nor overfitting. We tested this parameter range from 24 to 2016 (as shown in Fig. 4.6 (a)), and the corresponding update period of the dynamic validation set is two hours to one week. Due to the strong periodicity, it can be seen that changes to this value do not have a significant impact on performance for the data set METR-LA.

Then we tested parameter C, which means how many GNNs we save in the GNN pool. The GNN pool can save more historical information than only saving one GNN. We tested varied sizes of GNN pool, as shown in Fig. 4.6 (b) and based on the nature of the data set, the change of this parameter will not have a huge influence on the performance in the current data set.

The online learning rate e is also an important parameter in our method which means the degree of model learning after the new samples arrive. Since the correlation structure remains stable globally, the purpose of the self-adaptation process is to update sub-graphs changes locally caused by concept drift. Therefore, the e should not be set too large. If it is set too large, the model will be at risk of overfitting. Fig. 4.6 (c) shows its influence.

Another parameter is the number of the online learning epoch, which has a similar effect to parameter e (as shown in Fig. 4.6 (d)). It should be noted that we only need to adjust and update sub-graphs that contain correlation changes locally in our self-adaptation process. So this parameter should be set to a suitable value because if this parameter is set too large, the online prediction speed will be greatly reduced, which will go against the original intention of our online prediction method.

4.5 Summary

In this chapter, we propose a dynamic GNN-based self-adaptation method SAGN, which is designed for multi-stream under concept drift. Without using predefined graphs, we initialize a sampled correlation graph structure $\mathscr{G}_0$ based on historical data and it is embedded into GNN training. Multiple well-performed GNNs are stored in a GNN pool in the offline stage. When online testing starts, the dynamic adaptation process is performed by locally updating sub-graphs after new samples arrive. Meanwhile, globally the correlation graph structure of $\mathscr{S}$ will be preserved well. The designed dynamic GNN pool rolling mechanism cooperates with the dynamic validation set to periodically select the optimal GNN for the prediction task periodically. We tested the prediction performance of SAGN and baseline methods on real-world and synthetic multi-stream data sets. The experiment results show that our method performs the best in most cases.

CHAPTER 5

DYNAMIC GRAPH REGULARIZATION FOR MULTI-STREAM CONCEPT DRIFT SELF-ADAPTATION

5.1 Introduction

Concept drift, due to its potential to render models ineffective or lead to a decline in accuracy, has been a persistent challenge throughout the development of traditional machine learning [126]. It refers to the phenomenon where the data distribution changes over time [106]. For example, it could involve irregular changes in weather due to climate warming or shifts in public interest toward investment products caused by changes in the economic landscape. However, maintaining consistent data distribution between training and test data is crucial for traditional machine learning models to sustain performance [107]. Unfortunately, the concept drift problem is ubiquitous, especially in non-stationary data stream environments [127]. The widespread use of sensors, internet propagation, and the progress of modern society have led to the generation, update, and rapid change of streaming data in various areas. The unpredictability of concept drift,

with its unknown magnitude and duration, further exacerbates this challenge [128].

Self-adaptation offers a valuable approach to address the concept drift problem, enabling the model to learn and adapt to new data distributions [129]. This can be achieved through either an informed or blind strategy [53]. In the informed strategy, the model receives feedback from the drift detection module, and updates are made correspondingly when drift is unambiguously detected. Various statistical thresholds, such as error rates and multiple hypothesis tests, are integrated into the detection module to detect the concept drift [12]. On the other hand, the blind strategy entails a set of updating or learning rules [109], allowing the models to capture and learn changes in the data stream without relying on a drift detection mechanism [110].

A majority of the existing research on concept drift adaptation primarily focuses on single-stream tasks [111]. However, whether it's traditional forecasting tasks like weather and traffic or more complex ones like social media trends, solely considering the changes within individual data streams is no longer sufficient to meet real production demands [117]. Applying the single-stream adaptation method to multiple streams not only suffers from low prediction efficiency but also overlooks the differences and correlations between data streams [112]. For example, a real-world application involving real-time predictions on the load capacity of train carriages [130] necessitates managing multiple data streams. Establishing separate models for each route would consume significant computational resources and ignore the correlations between routes. In fact, factors contributing to drift, such as station control and temporary maintenance, are closely related to the connectivity and distance of the routes.

When we discuss data-driven tasks in multi-stream with considering the correlation between streams, each data stream is no longer isolated, which helps us further mine regionalized data information for different problems in a flexible task space [114]. In addition to the fixed correlation between streams, such as location information, the

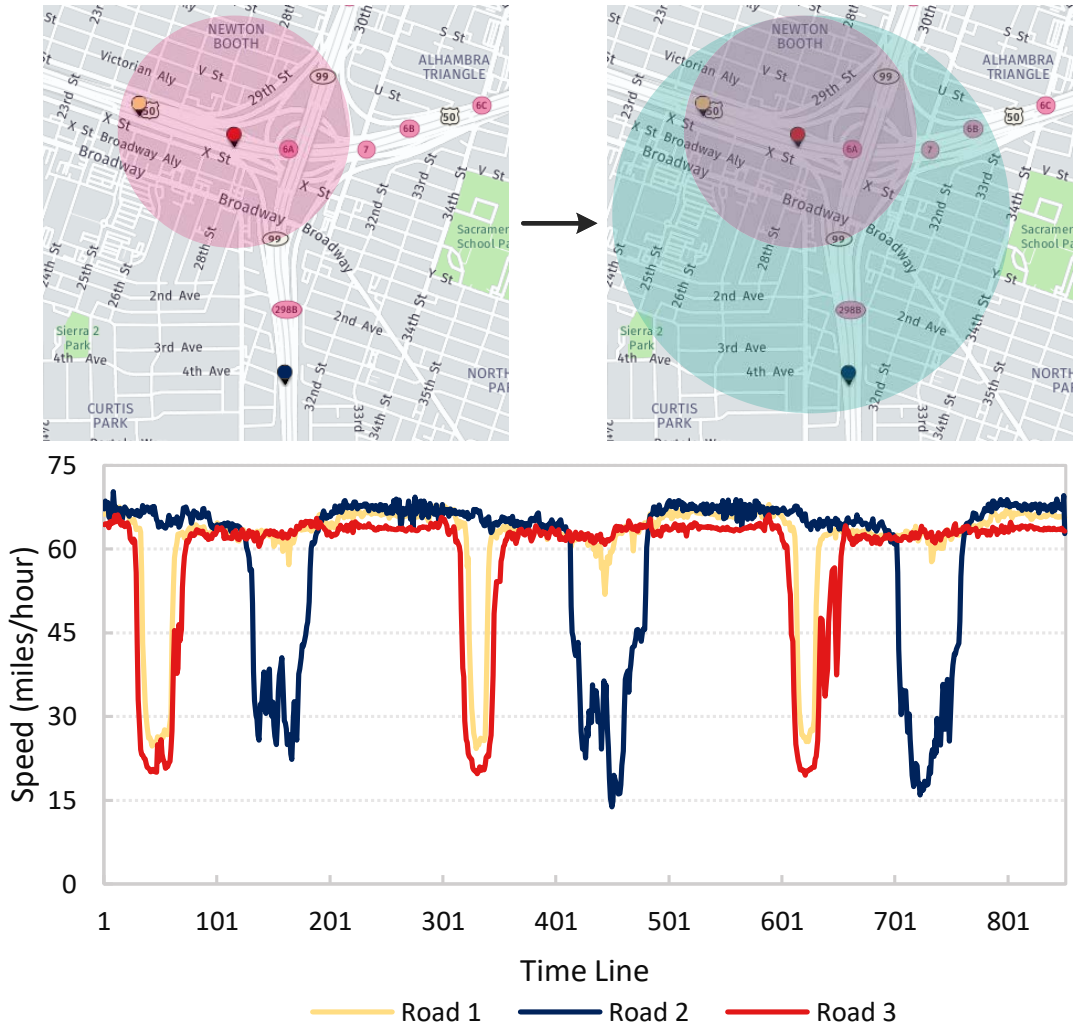


Figure 5.1: An example for traffic multi-stream correlation. Three sensors on three roads generate three data streams. The location information indicates the base correlation between streams. When a traffic accident occurs near Road 1 and Road 3, all roads in the red circle are affected first (concurrent drift), and then the roads in the blue circle are affected soon after (delayed drift).

correlation between drifts should not be overlooked. Drift-causing factors have the potential to impact one or more data streams, and they can be categorized as concurrent drift or delayed drift based on the time difference of their occurrence [111]. An example is shown in Fig. 5.1. The sensors on Roads 1, 2, and 3 generate three data streams. These sensors' locations represent the base correlation of multi-stream. When a traffic accident occurs near Road 1 and Road 3, all roads in the red circle are affected first (concurrent drift), and then the roads in the blue circle are affected soon after (delayed drift). Notably, the practical scenarios are complex; for example, if Road 3 is closed temporarily, then the base correlation will fail. Therefore, correlation mining for multi-stream environments becomes a huge challenge for self-adaptation tasks in the concept drift field.

However, current methods designed for multi-stream concept drift adaptation have several limitations. For instance, some are not suitable for large-scale datasets, while some require specific source and target streams to function effectively [116]. Moreover, certain approaches are limited to handling specific types of drift, which restricts their applicability [111]. Recently, some methods use graph neural networks (GNNs) to capture correlations between streams [88, 93, 131], but their static model fails to consider that the data distribution may change caused by concept drift in future data samples.

To fill these gaps, we propose a concept drift self-adaptation framework for multi-stream based on dynamic graph regularization, named Multi-stream Self-adaptation Graph Regularization (MSGR). MSGR can cover any drift type and is easy to apply to large-scale datasets. The adaptation process we design links the sub-graph updating with the degree of concept drift. Intuitively, the larger leaning weight should be assigned on the sub-graph updating when the drift is detected. In this chapter, we exploit graph structure and a graph neural network to capture the correlation among streams. When drift happens, which potentially indicates the change of this correlation, the graph structure and the network should be adjusted correspondingly. However, the correlation

graph structure learned in the offline training process might still be informative, and it is also a resource wasting to totally discard it. Thus, we propose a graph regularization to sub-graph updating for the graph neural network, which helps to capture the new correlation change caused by drifts.

Two aspects need to be considered when designing our framework: the performance of correlation graph structure construction and the dynamic response capability to the unknown drift. Specifically, for the former, we exploit a sampled graph and an adaptive matrix in our designed novel graph neural network to help capture deep spatial-temporal correlation among streams. Each node in the graph refers to a stream. This network learned a correlation graph structure based on the historical data distribution without any pre-defined graphs in the offline training stage. When the offline training process is completed, we use this GNN as the base prediction model for the multi-steam multi-step prediction task in the online testing. To detect the concept drift in the online testing stage, we use the loss of the base model on the validation set as the drift detection threshold. For the latter, to dynamically adapt to the distribution change on testing time, sub-graph updating with graph regularization and real-time loss monitoring are performed all the time. The sub-graphs are updated by newly multi-stream samples and the graph regularization weight is assigned dynamically according to the monitoring result. To calculate this weight, a dynamic kNN graph and a dynamic sampled graph are generated by newly arriving data in the sliding window to achieve this function.

This chapter's contributions are as follows:

- We propose a novel concept drift self-adaptation framework based on dynamic graph regularization for multi-stream, named MSGR. The graph-based regularization assigns dynamic learning weight for sub-graph updating according to the drift detection result, which can help the model adapt to the concept drift in multi-stream more effectively and targeted.

- A correlation graph structure is learned by our designed novel graph convolutional layer to capture the deep dependencies between streams. This graph structure is realized by embedding a sampling graph from a multi-stream historical data distribution and calculating an adaptive transition matrix based on each node pair.
- We conduct extensive experiments on three large-scale real-world datasets and synthetic datasets, and the results show that MSGR can achieve significant improvement over state-of-art baseline methods.

The work described in this chapter has been published in the IEEE-TKDE paper "Dynamic Graph Regularization for Multi-Stream Concept Drift Self-adaptation".

5.2 Problem Statement

In this section, the definitions and our problem statement are introduced. We present a summary of some important notations in Table 5.1 for brevity.

Definition 5.1 (Multi-Stream)**.** A multi-stream, denoted as $\mathscr{S}$, comprises several data streams $\mathscr{D}$, represented as $\mathscr{S} = \{\mathscr{D}^1, \mathscr{D}^2, \cdots, \mathscr{D}^i\}$. Each data stream $\mathscr{D}^i$ contains j feature time series, denoted as $\mathscr{D}^i = \{\mathscr{T}^1, \mathscr{T}^2, \cdots, \mathscr{T}^j\}$. The value of a specific time series $\mathscr{T}^j$ at time step t is represented as X_t^j.

The multi-stream environment is a significant challenge in the field of data mining. Our method is designed to handle the multi-series concept drift problem in multi-stream. Specifically, in our study, we consider that each data stream $\mathscr{D}^i$ in the multi-stream $\mathscr{S}$ contains the same feature time series. For example, multiple traffic monitor stations are placed in different areas of the city, and they monitor some same traffic features, such as speed, the number of vehicles, etc. using the sensors in these stations. Our research is based on the common features in these streams, where the same target feature series from multiple streams constitutes a multi-stream in our task.

Table 5.1: Summary of Notations

Notations	Description
$\mathscr{S} = \{\mathscr{D}^1, \cdots, \mathscr{D}^i\}$	A multi-stream contains i data streams.
$\mathscr{D} = \{T^1, \cdots, T^j\}$	A data stream contains j time series.
$\mathscr{T}^j = \{\mathrm{X}_t^j\}$	A time series in the data stream.
X^j	The feature value of the time series.
$t \in \mathbb{Z}^+$	The time step.
$\mathscr{P}$	The probability distribution.
Δ	The past window size.
η	The prediction time steps.
$\mathscr{G}$	The correlation graph structure of multi-stream.
$\mathscr{F}$	The predictor for multi-stream.
$\mathscr{S}^{tr}, \mathscr{S}^{val}, \mathscr{S}^{te}$	The training, validation and testing sets.

Definition 5.2 (The Correlation in Multi-Stream)**.** The correlation in multi-stream is denoted as $\mathscr{G}$. It comprises two parts, the correlation between streams and the correlation between concept drifts in different streams.

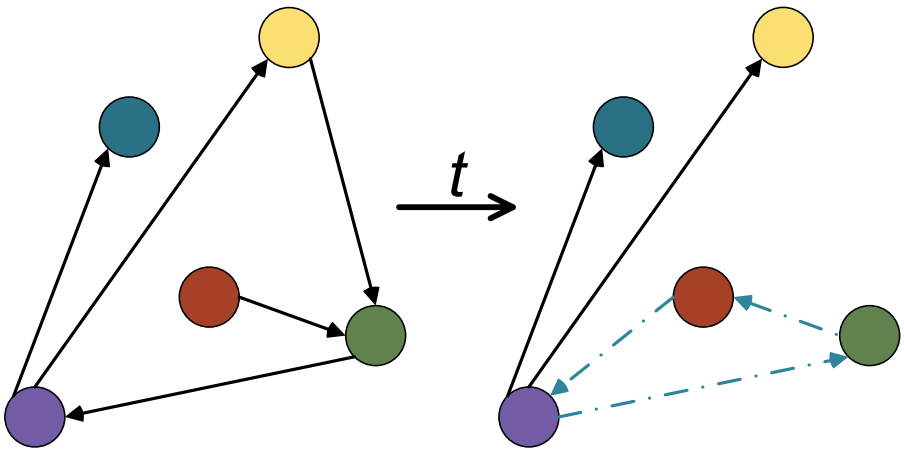


Figure 5.2: The Multi-Stream Correlation Changes Over Time.

As shown in Fig. 5.2, in a multi-stream, the correlation structure $\mathscr{G}$ is in a dynamic

state in streaming environments. The basic correlation between streams is usually formed by the location distance between sensors, so the global correlation structure will be stable in most cases, but sub-graphs will be dynamic. Concept drift is the main reason causing this local change.

For example, a traffic accident at a certain intersection may cause congestion in adjacent road sections, and thunderstorm occurrence in a local area will also cause a significant increase in the overall humidity in the area. On the other hand, if there is a parade or a road closure partly for repairs, the correlation will also change locally.

Definition 5.3 (Concept Drift in Multi-Stream)**.** Assume that $\mathscr{P}$ presents the data distribution. When $\mathscr{P}_{t+1}(\mathscr{S}) \neq \mathscr{P}_t(\mathscr{S})$, concept drift occurs at time step $t+1$. If concept drift occurs in any stream of $\mathscr{S}$, which is denoted as the concept drift in multi-stream.

Definition 5.4 (Concept Drift Correlation in Multi-stream)**.** Two cases of drift correlation exist in multi-stream based on the time difference. When concept drift occurs on more than one stream simultaneously, it is denoted as concurrent drift. If there is a time difference between these drifts, it is denoted as delayed drift. The detailed definition of the correlation between drifts in multi-stream is reported in [132].

Problem 3 (Adaptation Learning for Multi-Stream at t-step with Dynamic $\mathscr{G}_t$)**.** *Given previous δ-step information, and the correlation attributes $\mathscr{G}$, predict the next η-step for multi-stream, which can be formalized as*

$$\mathscr{F}_t, \mathscr{G}_t = \underset{f \in \mathscr{F}, \mathscr{G}}{\arg\min}\ \ell\left(f\left(\mathscr{S}_{t-\Delta+1:t}, \mathscr{G}_{t-1}\right), \mathscr{S}_{t+1:t+\eta}\right). \tag{5.1}$$

where $\mathscr{F}_t, \mathscr{G}_t$ are updated from $\mathscr{F}_{t-1}, \mathscr{G}_{t-1}$, this process is achieved by graph-based self-adaptation learning in the online testing stage.

The main difference between our task and traditional multi-series prediction tasks is that our task considers the concept drift problem in multi-stream during the testing

stage. This means our testing is conducted online, the prediction model is no longer static and it will be updated by a graph-based adaptation strategy dynamically based on newly arrived samples.

5.3 Methodology

This section describes the detailed architecture of our proposed MSGR. The framework of MSGR is shown in Fig. 5.3. Our method is divided into two stages: the offline training stage and the online testing stage. To obtain a more accurate correlation graph structure, a novel graph neural network is designed to learn the historical data distribution and deep correlation between streams in the training stage. In the online testing stage, to overcome the concept drift problem, we propose a graph-based dynamic regularization, which cooperates with sub-graph updates to realize self-adaptation. The real-time prediction error rate is monitored to detect the occurrence of drift, and it is connected to the regularization process to capture local correlation changes more accurately in the multi-stream.

5.3.1 Offline Initialization

Before online testing, we need to build a base prediction model, so we design a graph neural network to capture deep spatio-temporal correlations between streams. After training, we can obtain an initialised multi-stream correlation graph structure and a GNN model as the base prediction model for the online testing stage.

5.3.1.1 Graph Structure Convolutional Layer

The use of pre-defined graphs is a common approach to represent the correlation among multi-stream data. These pre-defined graphs typically include location information for sensors. However, pre-defined graphs may not always be available in practical scenarios

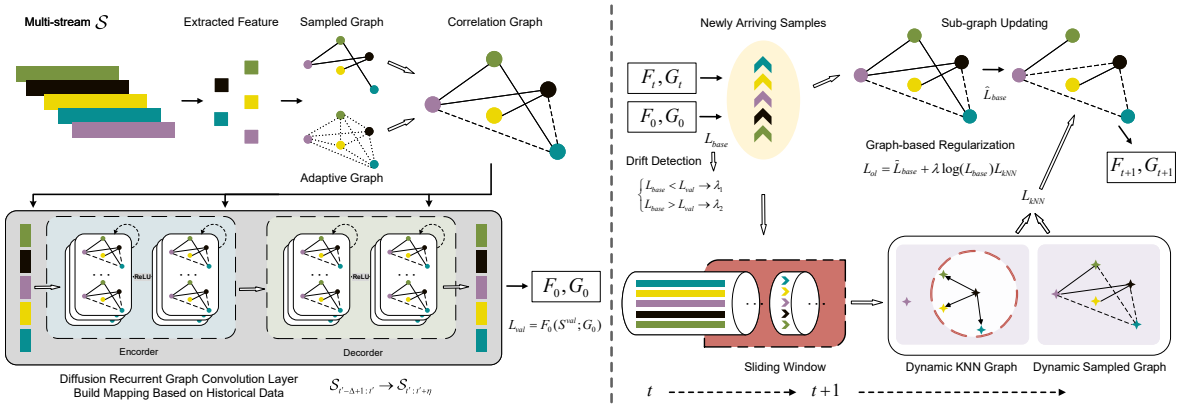


Figure 5.3: Overview of the MSGR framework. The training stage, depicted on the left, involves embedding the sampled graph and adaptive graph into the Diffusion Recurrent Graph Convolutional layer using a sequence-to-sequence structure. This stage aims to obtain a base prediction model. The image on the right shows our self-adaptation method during the online testing stage. The sub-graphs are updated using newly arriving data over time, and at the same time, dynamic graph-based regularization helps this updating process to adapt to the new data distribution. Two dynamic graphs generated by the new data stored in the sliding window achieve regularization weight calculation. The weight coefficient is determined based on the specific drift situation.

for multi-stream environments. Moreover, if we solely rely on fixed information from a pre-defined graph without ensuring a strong relationship with the prediction task, the prediction results may show significant deviations [95].

To learn more rich and deep correlation information from historical data of the multi-stream, we proposed a novel graph structure convolutional layer. Firstly, we simulate the historical data distribution of all streams based on sampling. However, a challenge exists here: the gradient cannot pass through the discrete graph structure after Bernoulli sampling. Motivated by [96], we employ the Gumbel-softmax trick [119, 120] to address this problem using a reparameterization process.

Specifically, a feature extraction network is utilized to process historical data in order to extract historical features and reduce data size. This involves applying convolutional layers to the entire training dataset $\mathscr{S}^{tr}$ along the time dimension, followed by vectorization (vec) of the convolutional (Conv) output. Subsequently, fully connected layers (FC)

are employed to further reduce the dimensionality, and each stream $\mathscr{S}^u$ in $\mathscr{S}^{tr}$ being mapped to a vector z^u. This process is represented as $z^u = \mathrm{FC}(\mathrm{vec}(\mathrm{Conv}(\mathscr{S}^u)))$.

After feature preprocessing, all data streams are combined in pairs of vectors (z^u, z^v). To obtain the pairwise combination of vectors, concatenation is performed by two fully connected layers: $\mu_{\langle u,v\rangle} \in [0,1]$, which gives the correlation weight between these two streams based on their distribution.

The reparameterization trick is employed here to calculate the sampled graph structure between every pair of streams u and v, denoted as G_{Gum}. Perturbed Gumbel noises based on uniform distribution are generated which correspond to N dimensional vectors Ω. $\{\mathrm{G}um_i\}_{i\leqslant N}$, the i.i.d sequence of standard Gumbel random variables can be calculated using the following equation:

$$\mathrm{G}um_i = -\log(-\log(U_i)), U_i \sim U(0,1). \tag{5.2}$$

Then we can get G_{Gum}:

$$G_{\mathrm{Gum}} = softmax(\frac{\log(\frac{\mu_{\langle u,v\rangle}}{1-\mu_{\langle u,v\rangle}}) + (\mathrm{G}um_u - \mathrm{G}um_v)}{\delta}) \tag{5.3}$$

where δ represents the temperature parameter.

To ensure the performance of the sampled graph is robust, we utilize a self-adaptive transition matrix [88] and embed it into the convolutional layer, which can dynamically learn node pair correlation probabilities constructed by matrix calculations to capture the deep correlation between streams in the convolutional process.

The self-adaptive transition matrix is optimized by two randomly initialized node embedding dictionaries with learnable parameters $E_1, E_2 \in \mathbb{R}^{N\times d}$:

$$A_{adp} = Softmax(ReLu(E_1 E_2^T)) \tag{5.4}$$

E_1 and E_2 are the source node embedding and target node embedding respectively, where each row of E_1 represents a node embedding and d refers to the embedding dimension. The spatially dependent weights between source nodes and target nodes are derived by matrix multiplication. The ReLu activation function is used to eliminate weak connections. The softmax function can normalize the adaptive adjacency matrix. Therefore, the normalized adaptive adjacency matrix can be considered as a transition matrix that hides the diffusion process. By embedding the sampled graph and self-learned adaptive graph, we obtain the following graph convolutional learning layer based on the spatial correlation:

$$\mathscr{G} = \sum_{p=0}^{P} G_{\text{Gum}}^{p} \mathscr{S} W_{p1} + A_{adp}^{p} \mathscr{S} W_{p2} \tag{5.5}$$

where P is the finite steps of the diffusion process of graph signals and W_p is the graph convolutional parameters of p-th order.

5.3.1.2 Temporal Convolutional Layer

In MSGR, we utilize a sequence-to-sequence structure to undertake the multi-step prediction mapping task: $S_{t-\Delta+1:t}$ to $S_{t+1:t+\eta}$. Our designed graph structure convolutional layer is embedded in a recurrent convolutional framework to complete spatio-temporal modelling and mapping tasks. At each time step t^*, the hidden state of the sequence-to-sequence model undergoes an update from h_{t^*-1} to h_{t^*}. The input for this update consists of the values of $\mathscr{S}_{t^*}$ within a defined historical window size Δ. Specifically, the encoder part of the model recurrently updates from $t^{'} - \Delta + 1$ to $t^{'}$, with h_{t^*} serving as the representation of $\mathscr{S}_{t^*}$. Subsequently, h_{t^*} is used to initialize the decoder. The hidden state then undergoes recurrent updates for each subsequent η-time step in the future, moving from $t^{'} + 1$ to $t^{'} + \eta$.

Many existing architectures can be applied here to serve this purpose. We use the Diffusion Convolutional Gated Recurrent Unit designed by [86] to complete the encoder-decoder function because it is more suitable for the directed graph structure. The diffusion process is achieved by a random walk on graph structure $\mathcal{G}$:

$$\Theta_\lambda \star_{\mathcal{G}} Z = \sum_{p=0}^{P} (\theta_{p,1}^{\lambda} (M_O^{-1}\mathcal{G})^p + \theta_{p,2}^{\lambda} (M_I^{-1}\mathcal{G}^T)^p) Z \tag{5.6}$$

where Θ is the filter to perform the diffusion convolutional operation on $\mathcal{G}$, M_O and M_I are the out-degree and in-degree matrix. Θ_λ for $\lambda = r, q, c$ are the model parameters for the corresponding filters. Then the graph diffusion module will replace the matrix multiplications in GRU [133]:

$$\begin{cases} r_{t^*} = sigmoid(\Theta_r \star_{\mathcal{G}} [\mathcal{S}_{t^*} \parallel \mathrm{h}_{t^*-1}] + b_r), \\ q_{t^*} = \tanh(\Theta_q \star_{\mathcal{G}} [\mathcal{S}_{t^*} \parallel (r_{t^*} \odot \mathrm{h}_{t^*-1}] + b_q), \\ c_{t^*} = sigmoid(\Theta_c \star_{\mathcal{G}} [\mathcal{S}_{t^*} \parallel \mathrm{h}_{t^*-1}] + b_c), \\ \mathrm{h}_{t^*} = c_{t^*} \odot \mathrm{h}_{t^*-1} + (1 - c_{t^*}) \odot q_{t^*} \end{cases} \tag{5.7}$$

where r_{t^*} and q_{t^*} are the reset gate and update gate at time step t^*. Mean absolute error (MAE) between the prediction values $\mathcal{Y} \in \mathbb{R}^{\eta \times I \times J}$ and ground truth $\hat{\mathcal{Y}} \in \mathbb{R}^{\eta \times I \times J}$ in future η time steps is the loss function of the training process, which can be defined as:

$$\mathcal{L}_{base} = \mathcal{L}(\hat{\mathcal{Y}}, \mathcal{Y}) = \frac{1}{\eta I J} \sum_{t=1}^{\eta} \sum_{i=1}^{I} \sum_{j=1}^{J} | \hat{\mathcal{Y}}_{ijt} - \mathcal{Y}_{ijt} |, \tag{5.8}$$

where I is the number of streams, J is the dimensionality of the output.

In the training stage, we construct a novel graph structure convolutional layer and utilize a new GNN-based network to map the spatio-temporal correlation among multi-stream data. Upon completion of the training, we can obtain a GNN model$\{\mathcal{F}_0, \mathcal{G}_0\}$, which

is used as the base prediction model for the multi-stream multi-step prediction task in the online testing stage. $\mathscr{G}_l$ includes the original correlation graph structure of multi-stream based on historical observations and contains two parts: global correlation and local correlation. Usually, the global correlation remains stable because the locations of the sensors are fixed. Local correlation refers to the associations between different data streams that arise due to various factors. The local correlation may change over time or due to the occurrence of concept drift. For example, in relation to roads in different areas of a city, roads that are in close proximity have similar busy degrees. These roads will suffer from similar traffic jam risks if a traffic accident occurs on one of the roads. But the local correlation changes caused by concept drift are not contained in the base prediction model, so an online-based prediction mode is necessary. Also, the loss of $\{\mathscr{F}_0, \mathscr{G}_0\}$ on the validation set $\mathscr{L}_{val}$ will be the threshold for concept drift detection in the testing stage.

5.3.2 Online Dynamic Graph-based Self-adaptation

The static prediction model carries a high risk when faced with unknown changes in streaming data. The concept drift means that new data distributions occur in newly arriving samples. To tackle this issue, we propose a novel self-adaptation strategy, achieved through dynamic graph-based regularization, to adjust the model over time continuously. This approach enables the timely capture and learning of any new unknown changes caused by concept drift, ensuring accurate multi-step prediction results are provided simultaneously.

We aim to stabilise the global correlation and adjust the model with accurate sub-graph updating. The work in [132] was the first to handle the concept drift problem in multi-stream in this way using simple online learning in cooperation with a GNN pool and a dynamic validation set.

But the shortcomings of this method are also evident. On the one hand, the multi-

Algorithm 2 MSGR dynamic self-adaptation process

1: Input Multi-stream $\mathscr{S}$, the input sequence length for all streams is Δ, and the output length is η. The batch size of the multi-stream in the testing stage is 1 (The new samples will arrive one by one). The online sub-graph updating rate is e_{ol}, the epoch number is Q_m. The loss function is calculated by MAE, and the optimization algorithm is Adam [122]. The k for dynamic kNN graph. The regularization weight coefficients are λ_1, λ_2 for specific drift situations. **2: Initial** The base prediction GNN model and correlation graph structure obtained from the training stage $\{\mathscr{F}_0, \mathscr{G}_0; \Theta_0\}$. The loss of $\mathscr{F}_0$ on the validation set is $\mathscr{L}_{val}$. The dynamic feature sliding window $\mathscr{V}^\tau$ is initialized by the historical data of $\mathscr{S}$, and the window size is τ.

for *New samples of* $\mathscr{S}$ *arrive in each time step,* $t = 1, 2, 3, \ldots$ **do**

3: Online Prediction: $\mathscr{F}_t\left(\mathscr{S}^{te}_{t-\Delta+1:t}; \mathscr{G}_{t-1}, \Theta_{t-1}\right) = S^{pre}_{t+1:t+\eta}$;// Give online prediction results for all streams

if $t > 1$ **then**

4: Obtain Ground Truth: $S^{true}_{t:t+\eta-1}$;

5: Update Dynamic Feature Window New arriving samples are spliced with the data in $\mathscr{V}^\tau$, and the old data at the remote end of the window is discarded;

6: Calculate Initial Model Loss

$\mathscr{L}_{base} = loss(\mathscr{F}_0\left(S^{te}_{t-\Delta:t-1}; \mathscr{G}_0, \Theta_0\right), S^{true}_{t:t+\eta-1})$;

7: Calculate Current Model Loss:

$\hat{\mathscr{L}}_{base} = loss(\hat{F}_{t-1}\left(S^{te}_{t-\Delta:t-1}; \mathscr{G}_{t-1}, \Theta_{t-1}\right), S^{true}_{t:t+\eta-1})$;

8: Generate Dynamic kNN Graph and Sampled Graph:

$G^\tau_{k\text{NN}}$ (see Eq.(5.9), G^τ_{Gum} (see Eq.(7.3));

9: Calculate Dynamic Regularization Loss:

$\mathscr{L}_{k\text{NN}}$ (see Eq.(5.10));

for $Q = 1, 2, \ldots Q_m$ **do** // Sub-graphs Updating under Dynamic Graph-based Regularization

if $\hat{\mathscr{L}}_{base} < \mathscr{L}_{val}$ **then**

10: Update under No Drift

$\mathscr{G}_{t-1}, \Theta_{t-1} = \text{Adam}(\mathscr{L}_{ol}; \lambda_1(\text{Eq.}(5.13)), F_t)$;

$\mathscr{G}_t = \mathscr{G}_{t-1}$, $\mathscr{F}_t = \mathscr{F}_{t-1}$;

else

11: Update under Concept Drift

$\mathscr{G}_{t-1}, \Theta_{t-1} = \text{Adam}(\mathscr{L}_{ol}; \lambda_2(\text{Eq.}(5.13)), F_t)$;

$\mathscr{G}_t = \mathscr{G}_{t-1}$, $\mathscr{F}_t = \mathscr{F}_{t-1}$;

end

end

end

end

model mechanism can partially avoid incorrect updates, but it consumes a considerable amount of storage resources, and the update process still lacks effective constraints. This means that it cannot adjust the degree of model updating based on the specific drift situation. On the other hand, many parameter settings dependent on the nature of the data also limit its applicability.

In our framework, these shortcomings have been addressed. Firstly, our proposed new graph neural work can learn deep correlations between data streams during training, and the high-performance initial prediction model eliminates the need to store multiple GNN models. Secondly, the proposed graph-based dynamic regularization is applied to the sub-graph updating process based on different drift situations, effectively adapting the new data distribution. Meanwhile, real-time error rates are linked to the regularization computation to capture local correlation changes accurately. Moreover, the simplified parameter settings significantly enhance the model's applicability.

Our designed regularization will take full advantage of the original graph structure. We use the newly arriving multi-stream data stored in a sliding window $\mathcal{V}^{\tau}$ with size τ to generate a dynamic weighted directed kNN graph and a dynamic sampled graph separately. The computational process to generate a sampled graph is the same as it is in the training stage. The dynamic kNN graph can be defined as:

$$G^{\tau}_{k\mathrm{NN}} = top_k f_{cos}(S^{\tau}), \tag{5.9}$$

where S_{τ} is the data in the sliding window, and k is the parameter of the kNN graph. $G_{k\mathrm{NN}}$ is updated over time with streaming data updating. The cross-entropy between the sampled graph and the dynamic kNN graph achieves the regularization function:

$$\mathscr{L}_{k\mathrm{NN}} = \sum_{uv} -G^{\tau}_{k\mathrm{NN}} \log G^{\tau}_{\mathrm{Gum}} - (1 - G^{\tau}_{k\mathrm{NN}}) \log(1 - G^{\tau}_{\mathrm{Gum}}) \tag{5.10}$$

Then the new loss function for the online sub-graph updating process can be defined as:

$$\mathscr{L}_{ol} = \hat{\mathscr{L}}_{base} + \mathscr{L}_{k\text{NN}}, \tag{5.11}$$

where $\hat{\mathscr{L}}_{base}$ is the loss calculated by the current model $\{\mathscr{F}_{t-1}, \mathscr{G}_{t-1}\}$ using samples in the window.

But applying the regularization at the same weight on the original loss function for the whole learning process is unreasonable, as inappropriate regularization will have a negative impact on the model. We design a dynamic regularization magnitude adjustment mechanism to address this problem, where λ is the weight coefficient for regularization depending on the drift detection result. The loss of the base model on the validation set $\mathscr{L}_{val}$ is used as the threshold for drift detection. The real-time error rate $\mathscr{L}_{base}$ calculated by the base model $\{\mathscr{F}_0, \mathscr{G}_0\}$ on the data in the window is monitored over time. When $\mathscr{L}_{base}$ exceeds $\mathscr{L}_{val}$, that indicates that the concept drift has occurred. According to the detection result, different weight coefficients λ_1, λ_2 are applied for no drift and drift detected situations, respectively. The online loss function can be defined as follows:

$$\mathscr{L}_{ol} = \hat{\mathscr{L}}_{base} + \lambda \mathscr{L}_{k\text{NN}}, \lambda \in \{\lambda_1, \lambda_2\} \tag{5.12}$$

To capture the local correlation change more accurately, we also link the real-time prediction error $\mathscr{L}_{base}$ to the regularization calculation, then the complete online loss function is as follows:

$$\mathscr{L}_{ol} = \hat{\mathscr{L}}_{base} + \lambda \log(\mathscr{L}_{base}) \mathscr{L}_{k\text{NN}}, \lambda \in \{\lambda_1, \lambda_2\} \tag{5.13}$$

Overall, dynamic self-adaptation is accomplished by the sub-graph updating process with the dynamic graph-based regularization: $\{\mathscr{F}_{t-1}, \mathscr{G}_{t-1}\} \rightarrow \{\mathscr{F}_t, \mathscr{G}_t\}$. Our design realizes the deep connection between the concept drift self-adaptation mechanism and the specific drift degree, which makes the online learning process more targeted and largely retains the model performance trained via historical information.

5.3.3 The MSGR Algorithm (Self-adaptation Stage)

The pseudo-code of our designed MSGR dynamic self-adaptation process is shown in Algorithm 2. In Step 1, the parameter setting for the online testing stage is conducted. In Step 2, the testing initialization is undertaken, where the base prediction model containing the correlation graph structure $\{\mathscr{F}_0, \mathscr{G}_0\}$ is used to perform the prediction task, and the loss of $\{\mathscr{F}_0, \mathscr{G}_0\}$ on the validation set $\mathscr{L}_{val}$ is the threshold of the drift detection. The online testing stage starts in Step 3, where new samples arrive at each time step and the model provides the nest η step prediction values of all streams. Our self-adaptation strategy, which involves sub-graph updating and graph-based regularization, is implemented to address the concept drift problem. Specifically, the ground truth is obtained in Step 4. Then, the newly available samples are stored and spliced in the sliding feature window in Step 5.

Steps 6 to 8 use the samples stored in the window to calculate different loss values and generate different dynamic graphs for regularization. In Step 6, the real-time error $\mathscr{L}_{base}$ is calculated by the original base model $\{\mathscr{F}_0, \mathscr{G}_0\}$. The model $\{\mathscr{F}_{t-1}, \mathscr{G}_{t-1}\}$ updated from the previous time step calculates the loss value $\hat{\mathscr{L}}_{base}$ in Step 7. Then in Step 8, a dynamic kNN graph and a sampled graph are generated separately. The cross-entropy $\mathscr{L}_{kNN}$ between these two graphs is calculated in Step 9. We apply the different weight coefficients λ_1 and λ_2 on the regularization to no drift and drift detected situations and the real-time error $\mathscr{L}_{base}$ is also linked to the regularization process. When $\mathscr{L}_{base} < \mathscr{L}_{val}$,

this indicates that there is no concept drift has occurred. Then the sub-graph updating will be performed using $\mathscr{L}_{ol}$ under λ_1 in Step 10. When $\mathscr{L}_{base} < \mathscr{L}_{val}$, this indicates that the concept drift has occurred, and the updating process in Step 11 will be performed under λ_2. Then the self-adaptation process of this time step $\{\mathscr{F}_{t-1}, \mathscr{G}_{t-1}\} \rightarrow \{\mathscr{F}_t, \mathscr{G}_t\}$ is completed.

5.4 Experiments

We construct extensive experiments to verify the effectiveness of the proposed MSGR. The real-world datasets (traffic and weather areas) are used to demonstrate the prediction performance of our MSGR by comparing it with existing frameworks. The synthetic datasets are used to further verify the self-adaptation performance by testing the occurrence of sudden drift in multi-stream.

We first give detailed information of the datasets and baseline models used for the experiments, and then the related settings, experimental results, and ablation study are discussed in corresponding subsections.

5.4.1 Datasets and Baselines

Datasets. Following previous works [88, 96, 132], our main experiments are constructed on three large-scale real-world datasets from the traffic and weather areas. A summary of these datasets is shown in Table 5.2.

Table 5.2: Summary of Datasets

Dataset	METR-LA	PEMS-BAY	WEATHER	SYNTHETIC	
				$\mathscr{S}^a$	$\mathscr{S}^b$
Nodes	207	325	10	10	10
Samples	34272	52116	29639	30000	30000
Update Rate	5min	5min	1h	5min	5min

- **METR-LA** is a dataset that captures the speed of traffic on the road network in LA County. The data is collected from loop detectors placed at 207 locations [123]. The dataset covers a duration of 4 months, from March to June in 2012 [86]. Traffic information is recorded at 5-minute intervals, resulting in a total of 34,272 time slices.

- **PEMS-BAY** is a publicly available dataset that captures traffic speed information from the Performance Measurement System (PeMS) of the California Transportation Agencies (CalTrans) [134]. It encompasses data from 325 sensors located in the Bay Area and spans a duration of 6 months, from January 1st, 2017 to May 31st, 2017 [86]. The traffic information is recorded at 5-minute intervals, resulting in a total of 52,116 time slices.

- **WEATHER** is a publicly available collection of weather data obtained from ten weather stations in Beijing, China [125]. It contains hourly weather measurements recorded from 3:00 AM to 3:00 PM, covering a time period from January 1st, 2015 to May 31st, 2018, totaling 1188 days. All weather stations share the same set of weather monitoring features. In our experiments, two specific weather features, namely $psur$ (surface air pressure in degrees celsius) and $q2m$ (specific humidity at a height of two meters above the ground in grams per kilogram), were selected.

- **SYNTHETIC DRIFT** is a synthetic data set generated by the Python package $scikit-multiflow$ in [132]. It contains two multi-streams based on different correlation settings. $\mathscr{S}^a$ has no correlation between streams. $\mathscr{S}^b$ is derived from the PEMS-BAY dataset that streams are related to each other. The subscript lir, tri, and pol represent the different drift functions, which are defined as follows:

$$\begin{cases} \mathscr{S}^*_{lir} = 5\mathscr{S}^* + 50 \\ \mathscr{S}^*_{tri} = \sin(5\mathscr{S}^*) + 50 \\ \mathscr{S}^*_{pol} = 0.01(\mathscr{S}^*)^{2.5} - 0.1(\mathscr{S}^*)^2 + 0.5\mathscr{S}^* \end{cases} \tag{5.14}$$

Then the datasets used for testing are $\mathscr{S}^a_{lir}$, $\mathscr{S}^a_{tri}$, $\mathscr{S}^a_{pol}$, and $\mathscr{S}^b_{lir}$, $\mathscr{S}^b_{tri}$, $\mathscr{S}^b_{pol}$.

Baselines. We select a wealth of baselines that contain traditional machine learning methods, typical GNN-based methods, and self-adaptation methods designed for multi-stream.

Traditional Methods:

- **HA**: Historical Average, an approach that utilizes a weighted average of past data to perform the prediction task;
- **VAR**: Vector Auto-Regression model;
- **SVR**: Support Vector Regression, which uses a linear support vector to achieve the regression aim;
- **FC-LSTM**: Fully Connected LSTM, indicating that the recurrent neural network is linked to the hidden units of the LSTM modules.

Typical GNN-based Methods:

- **DCRNN** [86]: The Diffusion Convolutional Recurrent Neural Network replaces the fully connected layer in the GRU [133] with a diffusion convolutional layer to model spatiotemporal dependency;
- **Graph-WaveNet** [88]: Graph WaveNet utilizes node embedding to learn an adaptable dependency matrix in the graph convolutional layer, which enables the model to capture both temporal and spatial dependencies effectively;

- **AGCRN** [95]: The Adaptive Graph Convolutional Recurrent Network incorporates two adaptive blocks to capture dependencies between streams: node adaptive learning and data-adaptive graph;
- **DGCRN** [91]: Dynamic Graph Convolutional Recurrent Network integrates dynamic graphs generated by dynamic filters and pre-defined graphs into a dynamic convolutional recurrent network to perform the prediction task;
- **ASTGCN** [90]: Attention-based Spatial-Temporal Graph Convolution Networks introduce a spatial-temporal attention mechanism that works in conjunction with graph convolution;
- **STFGNN** [89]: Spatial-Temporal Fusion Graph Neural Networks is a data-driven approach to generate temporal maps during training to compensate for the deep correlations that spatial maps may not reflect;
- **GTS** [96]: Graph for Time Series utilizes Gumbel sampling to generate the discrete graph structure among multiple data streams;
- **S^2TAT** [135]: Synchronous Spatiotemporal Graph Transformer simultaneous modelling of data by integrating attention mechanisms and graph convolutions in the proposed S^2TAT block.

Self-adaptation Methods Designed for Multi-stream:

- **MuNet** [111]: A concept drift self-adaptation framework achieved by combining a base predictor and Bayesian connectors. The algorithm can only be applied to small-scale datasets, so we choose ten streams for the traffic datasets for testing;
- **SAGN** [132]: A GNN-based self-adaptation framework for multi-stream under concept drift, performed by sub-graph online learning and the periodic validation and reselection of models stored in a GNN pool.

5.4.2 Experimental Setup

Platform. The model implementation is carried out using PyTorch 1.7.1. All experiments are conducted on a server equipped with an Nvidia Quadro RTX 8000 GPU with 48GB memory. The server's CPU is an Intel(R) Xeon(R) Gold 6226R CPU operating at 2.90GHz.

Parameter settings. In the training stage, the temperature parameter δ of Gumbel-softmax trick is set to 0.9. The training epoch number is set to 200 and the initial learning rate is set to 0.005 for the two traffic datasets and 0.001 for WEATHER. In the online testing stage, the sub-graph updating rate e_{ol} is set to 0.00001 for PEMS-BAY and 0.000005 for the other datasets. The sub-graph updating epoch number is set to 3 for all datasets when no drift occurs, and 5 for METR-LA under drift. The window size of the proposed dynamic KNN graph is set to 48 for METR-LA and PEMS-BAY, and 36 for WEATHER. The regularization penalty coefficient λ is set to 0.01 for the non-drift situation and 1 for the drift situation. The k for the KNN graph is set to 10 for METR-LA and PEMS-BAY, and 5 for WEATHER.

Evaluation Metrics. We use mean absolute error (MAE), mean absolute percentage error (MAPE) and root mean square error (RMSE) to evaluate the performance of MSGR and all the compared methods. The prediction values and ground truth values are transformed into the original values first and then the evaluation computation process is performed. These three evaluation matrixes are defined as follows [91]:

$$
\begin{aligned}
MAE &= \frac{1}{\eta}\sum_{t=1}^{\eta}|\mathscr{Y}_t - \hat{\mathscr{Y}}_t|, \\
MAPE &= \frac{1}{\eta}\sum_{t=1}^{\eta}\left|\frac{\mathscr{Y}_t - \hat{\mathscr{Y}}_t}{\mathscr{Y}_t}\right| \times 100\%, \\
RMSE &= \sqrt{\frac{1}{\eta}\sum_{t=1}^{\eta}(\mathscr{Y}_t - \hat{\mathscr{Y}}_t)^2}
\end{aligned}
\tag{5.15}
$$

where η is the number of prediction steps, $\mathscr{Y}_t$ denotes the ground truth and $\hat{\mathscr{Y}}_t$ represents the predicted values.

Table 5.3: The Performance Comparison between MSGR and the Baseline Methods on Real-world Datasets

	Time Step	Metric	HA	VAR	SVR	FC-LSTM	DCRNN	Graph WaveNet	AGCRN	DGCRN	ASTGCN	STFGNN	GTS	S^2TAT	MuNet	SAGN	***MSGR***
METR-LA	Step 3	MAE	4.16	4.42	3.99	3.44	2.77	2.69	2.87	2.62	4.86	3.23	2.64	2.78	10.02	2.46	**2.42**
		RMSE	7.80	7.89	8.45	6.30	5.38	5.15	5.58	5.01	9.27	7.43	4.95	5.43	10.02	4.41	**4.34**
		MAPE	13.00%	10.20%	9.30%	9.60%	7.30%	6.90%	7.70%	6.63%	9.21%	7.82%	6.80%	7.38%	15.73%	6.41%	**6.26%**
	Step 6	MAE	4.16	5.41	5.05	3.77	3.15	3.07	3.23	2.99	5.43	4.02	3.01	3.10	10.06	2.79	**2.72**
		RMSE	7.80	9.13	10.87	7.23	6.45	6.22	6.58	6.05	10.61	9.49	5.85	6.39	10.06	5.15	**5.04**
		MAPE	13.00%	12.70%	12.10%	10.90%	8.80%	8.37%	9.00%	8.02%	10.13%	10.00%	8.20%	8.70%	15.79%	7.64%	**7.41%**
	Step 12	MAE	4.16	6.52	6.72	4.37	3.60	3.53	3.62	3.44	6.51	5.05	3.41	3.43	10.13	3.16	**3.07**
		RMSE	7.80	10.11	13.76	8.69	7.59	7.37	7.51	7.19	12.52	11.67	6.74	7.32	10.13	5.91	**5.77**
		MAPE	13.00%	15.80%	16.70%	13.20%	10.50%	10.10%	10.38%	9.73%	11.64%	12.97%	9.90%	10.02%	15.97%	9.10%	**8.79%**
PEMS-BAY	Step 3	MAE	2.88	1.74	1.85	2.05	1.38	1.30	1.37	1.28	1.52	1.39	1.32	1.33	5.26	1.30	**1.29**
		RMSE	5.59	3.16	3.59	4.19	2.95	2.74	2.87	2.69	3.13	2.90	2.62	2.89	5.26	2.43	**2.43**
		MAPE	6.80%	3.60%	3.80%	4.80%	2.90%	2.73%	2.94%	2.66%	3.22%	2.96%	2.80%	2.83%	18.36%	2.75%	**2.71%**
	Step 6	MAE	2.88	2.32	2.48	2.20	1.74	1.63	1.69	1.59	2.01	1.77	1.64	1.62	5.31	1.60	**1.58**
		RMSE	5.59	4.25	5.18	4.55	3.97	3.70	3.85	3.63	4.27	3.97	3.41	3.73	5.31	3.11	**3.09**
		MAPE	6.80%	5.00%	5.50%	5.20%	3.90%	3.67%	3.87%	3.55%	4.48%	4.02%	3.60%	3.67%	18.46%	3.61%	**3.54%**
	Step 12	MAE	2.88	2.93	3.28	2.37	2.07	1.95	1.96	1.89	2.61	2.15	1.91	1.85	5.46	1.85	**1.81**
		RMSE	5.59	5.44	7.08	4.96	4.74	4.52	4.54	4.42	5.42	4.92	3.97	4.30	5.46	3.61	**3.55**
		MAPE	6.80%	6.50%	8.00%	5.70%	4.90%	4.63%	4.64%	4.43%	6.00%	5.17%	4.40%	4.31%	18.71%	4.37%	**4.26%**
WEATHER (psur)	Step 3	MAE	28.52	1.56	1.62	2.06	1.06	1.00	1.03	0.94	1.52	1.41	1.04	1.26	3.42	0.96	**0.93**
		RMSE	39.96	4.60	5.21	5.00	4.81	4.80	4.77	4.33	4.50	5.01	2.86	5.00	3.42	1.29	**1.25**
		MAPE	3.00%	0.16%	0.16%	0.21%	0.11%	0.10%	0.11%	0.10%	0.14%	0.14%	0.11%	0.13%	0.35%	0.10%	**0.09%**
	Step 6	MAE	28.52	2.30	2.57	2.86	1.51	1.44	1.48	1.56	2.15	2.01	1.55	1.83	3.45	1.43	**1.38**
		RMSE	39.96	5.36	6.29	5.80	5.23	5.13	5.11	5.09	5.32	5.32	3.39	4.74	3.45	1.79	**1.73**
		MAPE	3.00%	0.23%	0.26%	0.29%	0.16%	0.15%	0.15%	0.16%	0.22%	0.20%	0.16%	0.19%	0.35%	0.15%	**0.14%**
	Step 12	MAE	28.52	3.09	3.58	3.71	2.37	2.25	2.31	2.71	2.90	2.76	2.49	2.59	3.58	2.29	**2.21**
		RMSE	39.96	6.02	7.21	6.54	5.24	5.57	5.60	6.11	5.94	5.84	4.30	5.66	3.58	2.67	**2.56**
		MAPE	3.00%	0.31%	0.36%	0.38%	0.24%	0.23%	0.24%	0.28%	0.30%	0.28%	0.25%	0.26%	0.36%	0.23%	**0.23%**
WEATHER (q2m)	Step 3	MAE	5.15	0.61	0.25	0.30	0.59	0.56	0.59	0.56	0.61	0.61	0.60	0.63	1.33	0.56	**0.55**
		RMSE	6.29	1.04	0.58	0.58	1.03	1.00	1.03	1.01	1.02	1.04	0.88	1.12	1.33	0.69	**0.68**
		MAPE	191.12%	12.64%	13.32%	19.28%	11.20%	10.38%	11.20%	10.89%	13.39%	11.78%	11.26%	12.35%	42.04%	10.72%	**10.59%**
	Step 6	MAE	5.15	0.86	0.38	0.43	0.82	0.79	0.82	0.81	0.86	0.85	0.84	0.84	1.35	0.79	**0.77**
		RMSE	6.29	1.34	0.74	0.73	1.34	1.30	1.33	1.34	1.34	1.35	1.17	1.34	1.35	0.93	**0.91**
		MAPE	191.12%	18.84%	21.32%	29.07%	16.45%	15.30%	16.13%	16.61%	17.63%	16.94%	16.47%	16.10%	42.55%	16.02%	**15.60%**
	Step 12	MAE	5.15	1.17	0.53	0.59	1.16	1.10	1.16	1.17	1.16	1.14	1.16	1.15	1.55	1.10	**1.06**
		RMSE	6.29	1.72	0.89	0.88	1.77	1.70	1.74	1.79	1.72	1.73	1.53	1.74	1.55	1.25	**1.20**
		MAPE	191.12%	27.73%	32.70%	42.29%	24.98%	23.33%	25.00%	25.83%	25.11%	24.66%	24.43%	24.88%	46.65%	24.24%	**23.20%**

5.4.3 Experiment Results

We evaluate MSGR on three real-world datasets in different scales, as shown in Table 5.2, where the three datasets vary significantly in data scale. This demonstrates MSGR's adaptability to different-scale data.

Table 5.3 shows the overall comparison results. Our method achieves the best performance in multi-step prediction tasks across all datasets. The PEMS-BAY and WEATHER datasets exhibit more regular changes, while the METR-LA dataset experiences more drifts in the testing stage.

Remarkably, MSGR demonstrates a noteworthy average accuracy improvement over the state-of-the-art traditional GNN-based model by **9.14%** on the METR-LA dataset. Additionally, it outperforms the SAGN model, which incorporates concept drift adaptation, with an average improvement of **2.33%** on the METR-LA dataset. MSGR achieves significant improvements, particularly in far-step prediction tasks. Specifically, for the Horizon 12 prediction task, MSGR achieves an improvement of **11.07%** compared to the traditional GNN-based models and a **2.93%** improvement compared to the SAGN adaptation model.

MSGR accurately predicts results even in datasets (PEMS-BAY and WEATHER) with periodic changes, such as recurring drifts caused by seasons. It leverages the learned correlation graph structure from historical data and adapts to new minor changes in incoming data samples.

We select SAGN, which exhibits the best performance among the baseline methods, and conducted a statistical test of significance between it and our proposed MSGR based on three evaluation metrics. The results are shown in Table 5.4, indicating that our method significantly outperforms SAGN at the 0.05 significance level.

We also compare the runtime to further analyze the complexity of our method. We select a traditional GNN-based method, GTS, a GNN-based adaptation method, SAGN, and

Table 5.4: Statistical Test ($^*p < 0.05$)

SAGN	MAE	MAPE	RMSE
p-value	4.46×10^{-6}	3.7×10^{-4}	5.53×10^{-4}

our method for comparison. The result is shown in Table 5.5. All methods are performed offline and based on the same historical data for the training stage. As for the testing stage, even though our model runs in online mode and requires continuous updating and adjustment to achieve the self-adaptation process, our prediction time cost for each time step averages 1.06 seconds, which is much smaller than the dataset sample update rate.

Table 5.5: The Runtime Comparison

	Runing Time (METR-LA)	
Method	Training Stage (s/epoch)	Testing Stage (s/sample)
GTS	48.87	0.0006
SAGN	49.24	0.34
MSGR	51.96	1.06

Table 5.6: The Ablation Experiments

	Step	Metric	DGCRN	w/o ol	w/o adp	w/o λ	***MSGR***
METR-LA	Step 3	MAE	2.62	2.60	2.47	2.44	**2.42**
		RMSE	5.01	4.42	4.87	4.37	**4.34**
		MAPE	6.63%	6.86%	6.41%	6.30%	**6.26%**
	Step 6	MAE	2.99	2.97	2.80	2.74	**2.72**
		RMSE	6.05	5.78	5.16	5.07	**5.04**
		MAPE	8.02%	8.36%	7.69%	7.44%	**7.41%**
	Step 12	MAE	3.44	3.35	3.17	3.09	**3.07**
		RMSE	7.19	6.65	5.92	5.80	**5.77**
		MAPE	9.73%	10.07%	9.10%	8.83%	**8.79%**

5.4.4 Ablation Study

Table 5.7: The Self-adaptation Performance Comparison

		Step 3			Step 6			Step 12		
Dataset	Method	MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE
$\mathscr{S}^a_{lir}$	SAGN	5.40	6.65	2.73%	5.40	6.64	2.73%	5.43	6.69	2.75%
	MSGR	**4.96**	**6.19**	**2.42%**	**4.96**	**6.19**	**2.43%**	**4.98**	**6.22**	**2.44%**
$\mathscr{S}^a_{tri}$	SAGN	0.67	0.76	1.34%	0.67	0.76	1.34%	0.67	0.76	1.35%
	MSGR	**0.67**	**0.76**	**1.34%**	**0.67**	**0.76**	**1.34%**	**0.67**	**0.76**	**1.34%**
$\mathscr{S}^a_{pol}$	SAGN	0.62	0.74	3.47%	0.62	0.73	3.46%	0.62	0.74	3.49%
	MSGR	**0.54**	**0.68**	**3.06%**	**0.54**	**0.68**	**3.04%**	**0.54**	**0.68**	**3.04%**
$\mathscr{S}^b_{lir}$	SAGN	7.67	10.06	7.27%	9.11	12.48	9.02%	11.30	15.43	11.40%
	MSGR	**6.36**	**8.71**	**6.11%**	**8.06**	**11.28**	**7.99%**	**10.67**	**14.85**	**10.99%**
$\mathscr{S}^b_{tri}$	SAGN	0.95	1.10	1.78%	0.98	1.14	1.84%	1.01	1.16	1.86%
	MSGR	**0.83**	**0.98**	**1.65%**	**0.83**	**0.98**	**1.67%**	**0.86**	**1.01**	**1.72%**
$\mathscr{S}^b_{pol}$	SAGN	3.19	4.52	28.34%	3.67	5.37	39.07%	4.36	6.44	54.13%
	MSGR	**3.09**	**4.36**	**25.98%**	**3.60**	**5.25**	**37.36%**	**4.32**	**6.37**	**49.38%**

To verify the effectiveness of various components of our framework, we conducted ablation experiments on the METR-LA dataset.

w/o ol refers to testing in the offline mode, where both the sub-graph updating and dynamic graph-based regularization modules are removed. As shown in Table 5.6, our method still outperforms the existing methods even when operating completely in the offline mode. This highlights the significance of our novel graph convolutional layer in providing a superior base prediction model for the online testing stage.

w/o adp indicates that the model adjustments rely solely on sub-graph updating when removing the dynamic graph-based regularization module. It can be observed that, compared to the offline prediction mode, sub-graph updating is an effective adaptive approach. However, the rough updating method also limits the model's performance. So we design the regularization method to utilize the latest data features to construct

the graph structure-based regularization to aid the model in capturing local correlation changes more accurately.

w/o λ represents that the drift detection module is removed, which means the regularization is applied on the sub-graph updating process in the same weight during testing. The result shows that if the self-adaptation process can be targeted, the model performance can be improved further. And our method achieves the best performance in the testing.

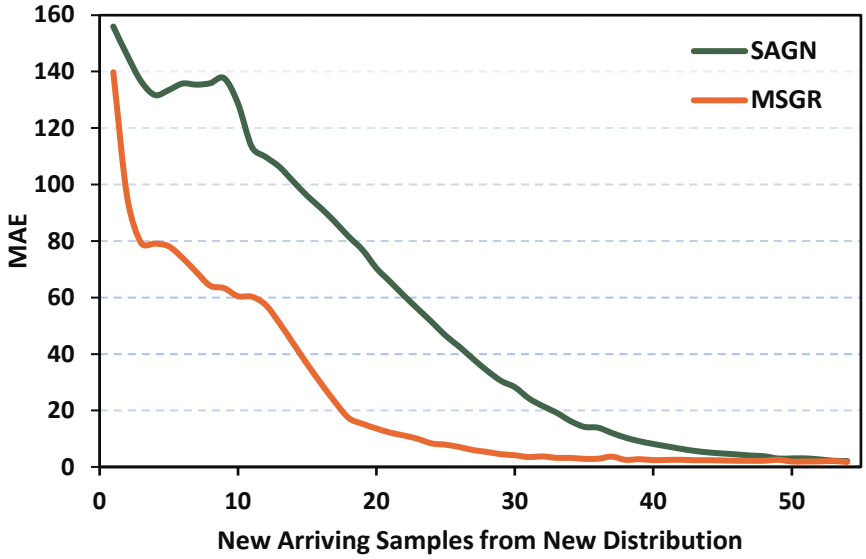
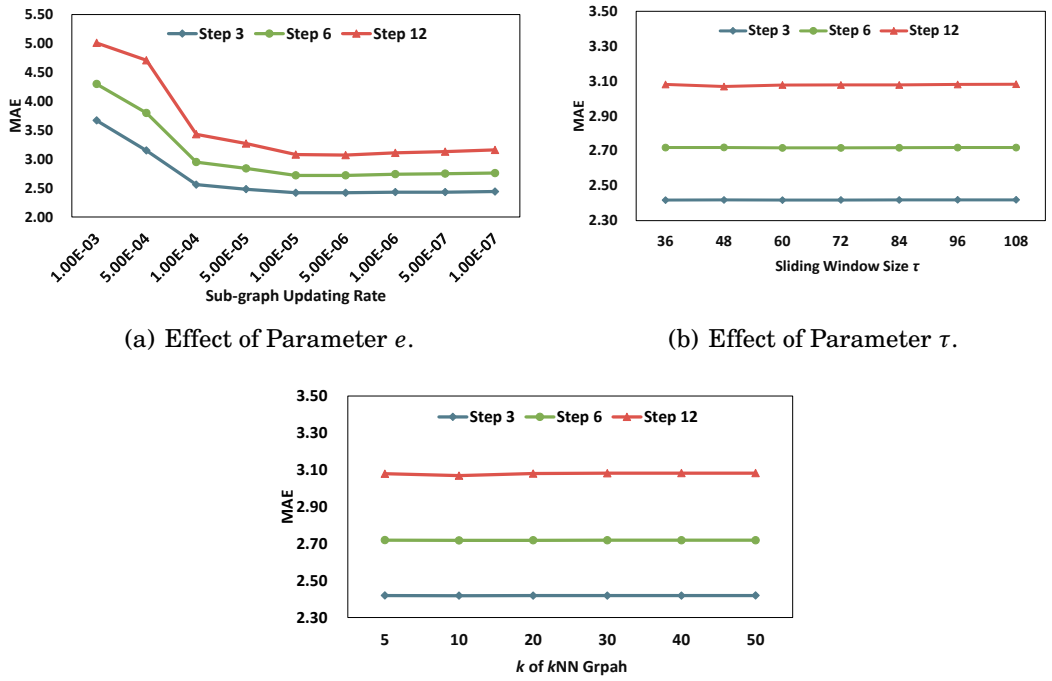


Figure 5.4: The Self-adaptation Performance Comparison.

To evaluate the self-adaptation performance of MSGR, we further test our method on SYNTHETIC DRIFT, including more pronounced concept drift situations [99]. The testing results are shown in Table 5.7. Compared with SAGN, our method achieves a better average prediction performance in most drift cases. Fig. 5.4 compares the self-adaptation speed (tested on $\mathscr{S}^b_{tri}$ dataset). When the multi-stream faces sudden drift, our method achieves faster adaptation and the model converges more efficiently.



(a) Effect of Parameter e.

(b) Effect of Parameter τ.

(c) Effect of Parameter k.

Figure 5.5: The effect of different parameters in the self-adaptation stage.

5.4.5 Parameter Sensitivity

In this section, we test the parameter sensitivity of our method. We test three parameters in our framework: the sub-graph updating rate e, the k for kNN graph and sliding window size τ for two dynamic graphs. The testing results are shown in Fig. 5.5.

The sub-graph updating rate e (Fig. 5.5 (a)), represents the degree of learning of the model for new samples. Due to the stable global correlation structure, this parameter cannot be too large, or the model will be at risk of overfitting.

The parameter τ is the size of the sliding window, and it determines how many recently obtained new samples are used to generate the kNN graph and the sampled graph. Using the latest samples as features to complete the regularization process helps the model learn and adapt correctly to the new data distribution. Through testing (Fig. 5.5

(b)), it can be seen that this parameter is not sensitive. However, considering the update rate of samples, setting this parameter too large may lead to an extension of the model update time.

The k represents the range of nearest neighbours for which we capture instantaneous local correlations with the kNN graph. We set this parameter to be a suitable value according to the specific situation of the practical dataset. It can be observed from Fig. 5.5 (c) that this parameter is not sensitive. Therefore, setting this parameter to a large value is not recommended to avoid excessive noise weighting.

5.5 Summary

In this chapter, we propose a novel graph structure self-adaptation framework, MSGR, which is designed for multi-stream under concept drift. Our framework first constructs a novel graph neural network architecture to capture deep section-temporal correlations between streams via historical data without pre-defined graphs. A high-performance GNN model is obtained after training and used as the base prediction model to perform the multi-stream multi-step task in online testing. Then, the sub-graph updating and our proposed dynamic graph-based regularization complete the self-adaptation process. The regularization weight coefficient is dependent on the drift detection results, meaning that a larger regularization weight corresponds to cases of drift. This mechanism adjusts and updates the base model to adapt to concept drift in multi-stream. Any distribution and correlation changes are captured in time and accurate prediction results of all streams will be given simultaneously. Detailed and comprehensive experiments demonstrate the effectiveness and superior performance of MSGR. Specifically, MSGRãchieves an average accuracy improvement of **9.14%** over state-of-the-art traditional GNN-based models and **2.33%** over the SAGN model with concept drift adaptation on the METR-LA dataset, with particularly notable gains in far-step prediction tasks.

CHAPTER 6

CONTINUOUS GRAPH LEARNING-BASED SELF-ADAPTATION FOR MULTI-STREAM CONCEPT DRIFT

6.1 Introduction

In stream environments, the phenomenon of data distribution changing over time is known as concept drift [12]. This unpredictability poses a significant challenge in stream environments [4]. Concept drift is common and inevitable, manifesting in various practical scenarios, such as changing popular topics on social media or sudden adverse weather conditions [56]. Traditional machine learning methods often rely on the consistency of data distribution between training and test sets [99]. These models perform well when the training data accurately represents the test data, but fail when faced with new, unseen data distributions, leading to a substantial drop in performance [107]. With the rapid increase in data stream updates due to the proliferation of sensors, static models based solely on historical data are insufficient for handling complex real-time prediction tasks [136].

To address the concept drift problem, researchers have developed numerous frameworks for single-stream environments using various adaptation strategies, enabling models to learn from new data distributions [137]. These frameworks often rely on drift detection modules or pre-set rules to trigger the model updates [14]. While these methods mitigate performance loss caused by concept drift to some extent, they have notable limitations [114]. One significant limitation is scalability. Although single-stream approaches can serve as a starting point for tackling concept drift, more complex practical tasks often involve multi-stream applications, such as multi-site weather forecasting or real-time monitoring of road traffic networks [112]. For instance, real-time predictions on the load capacity of train carriages require managing multiple data streams [130]. Applying self-adaptation methods to each stream separately in a multi-stream setting can be a potential solution, but it is inefficient. This approach ignores the correlations between streams and leads to redundant computations [111]. Consequently, developing effective solutions for multi-stream environments has become a significant challenge in the field of concept drift adaptation [19].

The complexity of the multi-stream surpasses that of the single-stream due to not only the increased number of data streams but also the necessity to consider more intricate correlations between streams [138]. For instance, alterations in the correlation among streams may arise in instances of drift. As illustrated in Fig. 6.1, two bridges connect the two sides of the bay. When the roads are clear, the black arrows in the top image indicate both directions are passable. However, if a traffic accident occurs at the location of sensor A, causing one of the bridges to be temporarily impassable, the nearby road segment (sensor B) will be immediately affected, which we refer to as concurrent drift. At the same time, since the bridge near sensor A is currently blocked, vehicles needing to cross the bay will have to take a detour. Consequently, a large number of vehicles will converge at the entrance of the other bridge via the route indicated by the red arrow

in the bottom image. This will result in a traffic jam at the location of sensor C, which we refer to as delayed drift. Hence, understanding the correlations between streams is crucial for constructing robust models that excel in multi-stream environments.

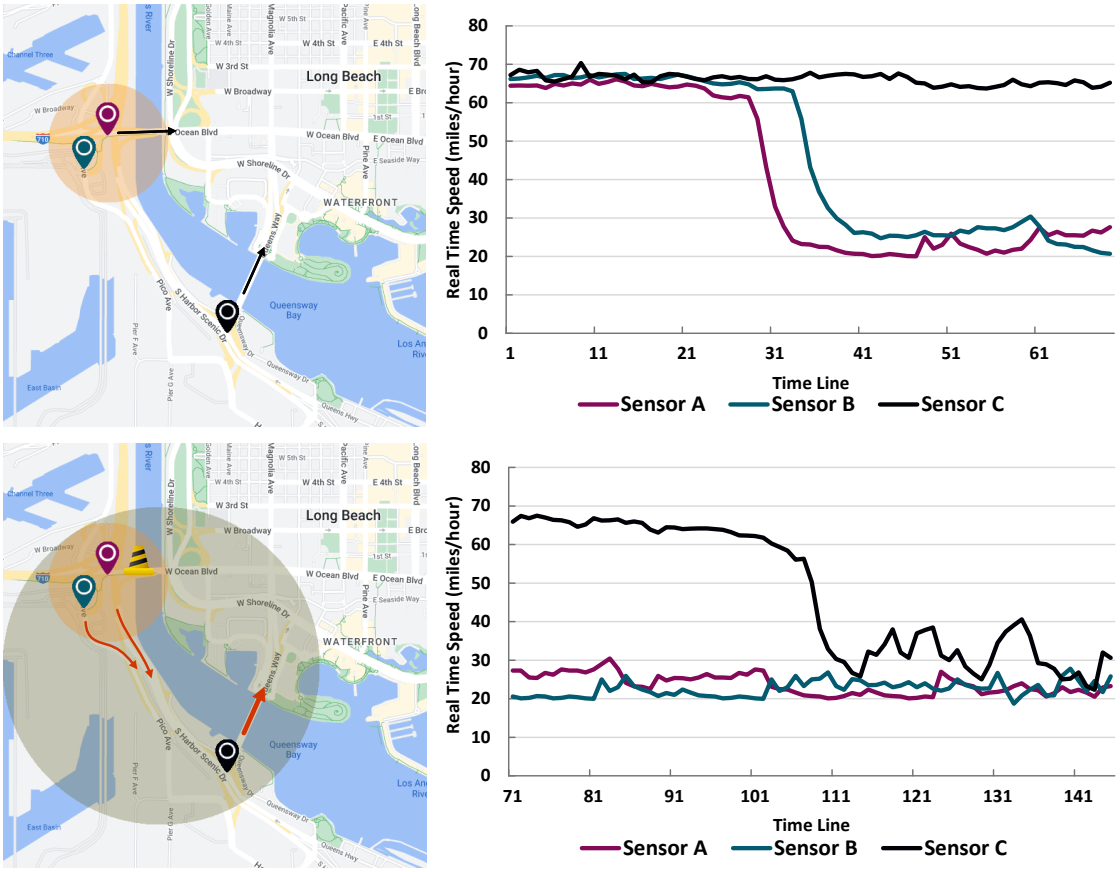


Figure 6.1: An example of traffic multi-stream correlation change. Each sensor generates a data stream. The positional relationship of the three sensors forms the fundamental correlation. However, the original correlation will change due to drift occurrences (such as traffic accidents).

Most existing methods crafted for multi-stream scenarios face limitations that hinder their direct applicability [79]. For instance, some methods are designed under the data scale requirements, allowing them to handle only two to three streams simultaneously. Others necessitate specific source and target streams to initiate processing [116], while some are tailored to address particular types of drift [111]. While these methods offer insights into addressing concept drift in multi-stream settings to a certain degree, they

remain inadequate. Recently, contemporary researchers have been exploring graph-based structures to capture inter-stream correlations [139]. Although these structures may not directly tackle the concept drift problem, they offer a fresh perspective for advancing multi-stream concept drift research.

To address these gaps, we propose a continuous graph learning-based self-adaptation framework for multi-stream concept drift, named CGLM. Our CGLM is suitable for large-scale multi-stream data and can adapt to any form and degree of concept drift. In designing CGLM, we focus on two key aspects: the model's initialization conditions and its dynamic adaptability to drift. Our framework, based on graph neural networks, aims to represent the complex spatio-temporal relationships between multiple streams using graph structures. However, in real-world scenarios, pre-defined graphs may not always be available for initializing the correlation graph structure, and the amount of data available for model training may be limited. This challenge prompts us to design a new graph neural network architecture embedded with an Adaptive Graph Generator (AGG). In situations with limited historical data and no pre-defined graphs, AGG can generate an adaptive correlation graph structure to aid in model initialization. Moreover, the model initialized with AGG exhibits better performance than those initialized with pre-defined graphs.

When the testing phase begins, we use the online mode to simulate the continuous updating of the multi-stream. The initially obtained base prediction model performs the multi-step prediction task for all streams. We monitor the real-time performance of the model to detect drift. Online self-adaptation relies on sub-graph updating, with different continuous graph learning mechanisms triggered based on drift detection. In non-drift scenarios, sub-graph updating is achieved by lightweight weight adjustments based on a few newly arriving samples. When drift occurs, AGG first generates a new dynamic graph based on new samples stored in the sliding window. The designed Adaptive Diffusion

Graph Attention Module (ADGAT) then captures the local correlation changes caused by the drift. Specifically, ADGAT treats the new graph as the feature graph at the current time step, applies a multi-head diffusion attention mechanism, and adaptively updates the sub-graphs of the original correlation graph structure. Furthermore, the degree of drift is linked to the sub-graph updating rate to achieve more accurate dynamic self-adaptation.

The contributions of this chapter are as follows:

- We propose CGLM, a continuous graph learning-based self-adaptation framework for handling multi-stream concept drift. We design a novel graph convolutional layer embedded in an Adaptive Graph Generator (AGG). This approach captures deep spatio-temporal relationships between streams without pre-defined graphs. We employ distinct continuous graph learning mechanisms tailored for both non-drift and drift scenarios, ensuring high-performance dynamic self-adaptation to concept drift.

- The Adaptive Diffusion Graph Attention Module (ADGAT) is designed to capture local correlation changes for drift self-adaptation. Initially, AGG constructs a dynamic feature graph based on the newly arriving samples. Subsequently, ADGAT applies a multi-head diffusion attention mechanism to this feature graph, enabling adaptive sub-graph weight updating for the original correlation graph while maintaining global correlation stability.

- We conduct extensive experiments on three large-scale real-world datasets. The results show that when only small-scale data is available for model initialization, our proposed CGLM can achieve significant improvement over state-of-the-art baseline methods. This also demonstrates that CGLM maintains stable performance over long-term updates. Additionally, our method maintains leading prediction performance as the training data volume increases.

The work described in this chapter has been published in the IEEE-TCYB paper "Continuous Graph Learning-based Self-adaptation for Multi-stream Concept Drift".

6.2 Problem Statement

In this section, we first introduce the definitions and our problem statement. A summary of important notations is shown in Table 6.1 for brevity.

Table 6.1: Summary of Notations

Notations	Description
$\mathscr{S} = \{\mathscr{D}^1, \cdots, \mathscr{D}^i\}$	A multi-stream contains i data streams.
$\mathscr{D} = \{\mathscr{T}^1, \cdots, \mathscr{T}^j\}$	A data stream contains j time series.
$\mathscr{T}^j = \{\mathrm{X}_t^j\}$	A time series in the data stream.
X^j	The feature value of the time series.
$t \in \mathbb{Z}^+$	The time step.
$\mathscr{P}$	The probability distribution.
Δ	The past window size.
η	The prediction time steps.
$\mathscr{G}$	The correlation graph structure of multi-stream.
$\mathscr{F}$	The predictor for multi-stream.
$\mathscr{S}^{tr}, \mathscr{S}^{val}, \mathscr{S}^{te}$	The training, validation and testing sets.

Definition 6.1 (Multi-Stream)**.** A multi-stream, denoted as $\mathscr{S}$, comprises i data streams $\mathscr{D}^i$: $\mathscr{S} = \mathscr{D}^1, \mathscr{D}^2, \cdots, \mathscr{D}^i$. For each data stream $\mathscr{D}^i$ in the multi-stream $\mathscr{S}$, it contains j feature time series, denoted as $\mathscr{D}^i = \{\mathscr{T}^1, \mathscr{T}^2, \cdots, \mathscr{T}^j\}$. At time step t, the value of a specific time series $\mathscr{T}^j$ is denoted as X_t^j.

Developing frameworks adapted to multi-stream environments poses a significant challenge in data mining. Our method aims to tackle the multi-series concept drift problem

in multi-stream. In particular, our analysis assumes that the same feature time series are contained in each data stream $\mathscr{D}^i$ of the multi-stream $\mathscr{S}$. For instance, multiple road monitoring points treated as multiple streams are deployed in different areas of a city, and different sensors monitor many features in these points, such as speed, vehicle count, etc. Our research focuses on these common features across streams, where the same target feature series from multiple streams form a multi-stream in our task.

Definition 6.2 (The Correlation in Multi-Stream)**.** The correlation in multi-stream is represented as $\mathscr{G}$. When we discuss $\mathscr{G}$, in addition to the correlation between streams, the correlation between drifts also needs to be taken into account.

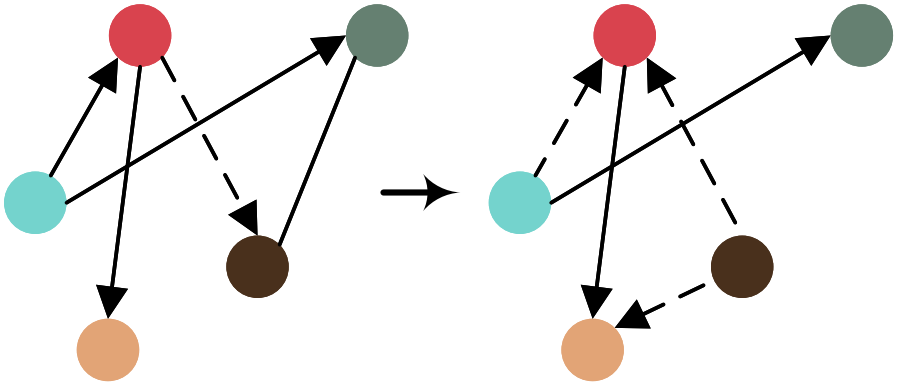


Figure 6.2: The Multi-Stream Correlation Changes Over Time.

As shown in Fig. 6.2, the correlation structure $\mathscr{G}$ of a multi-stream is dynamic in streaming environments. The basic correlation between streams is usually formed by the location distance between sensors, so the global correlation structure remains stable in most cases, but sub-graphs will be dynamic. Concept drift is the main reason for these local changes.

For example, in a city, if there's a traffic accident on a road, adjacent roads will be the first to be affected. Depending on the connectivity and proximity of the roads, congestion caused by the accident will gradually spread to a larger area. Similarly, localized thun-

derstorms in certain areas or fires in mountainous regions can also lead to changes in local correlations due to unknown factors causing drift.

Definition 6.3 (Concept Drift in Multi-Stream)**.** Let $\mathscr{P}$ represents the data distribution. Concept drift occurs at time step $t+1$ if $\mathscr{P}_{t+1}(\mathscr{S}) \neq \mathscr{P}_t(\mathscr{S})$. If concept drift occurs in any stream of $\mathscr{S}$, it is denoted as concept drift in the multi-stream.

Definition 6.4 (Concept Drift Correlation in Multi-stream)**.** In multi-stream, there are two instances of drift correlation depending on the time difference. Concurrent drift refers to concept drift that happens on many streams at the same time. Delayed drift refers to a time interval between these drifts. [132] provides a comprehensive definition of the correlation between drifts in a multi-stream.

Problem 4 (Self-Adaptation Learning for Multi-Stream at t-step with Dynamic $\mathscr{G}_t$)**.** *Predict the next η-step for multi-stream using the correlation attributes $\mathscr{G}$ and the prior δ-step information. It can be formalized as*

$$\mathscr{F}_t, \mathscr{G}_t = \underset{f \in \mathscr{F}, \mathscr{G}}{\arg\min}\ \ell\left(f\left(\mathscr{S}_{t-\Delta+1:t}, \mathscr{G}_{t-1}\right), \mathscr{S}_{t+1:t+\eta}\right). \tag{6.1}$$

where $\mathscr{F}_t, \mathscr{G}_t$ are updated from $\mathscr{F}_{t-1}, \mathscr{G}_{t-1}$, this process is achieved by continuous graph learning-based self-adaptation in the online testing phase.

The primary distinction between our task and traditional multi-series prediction tasks is that we address the concept drift problem in multi-streams during the testing phase. In other words, our testing is conducted online, where the prediction model is no longer static. Instead, it is dynamically updated through our designed continuous graph learning-based self-adaptation method.

6.3 Methodology

In this section, we introduce detailed information about the structure of our proposed CGLM. The overview of CGLM's framework is illustrated in Fig. 6.3. Our method operates in both the offline training and online testing phases. During the initialized training phase, we construct a novel graph neural network embedding an Adaptive Graph Generator (AGG) module to capture deep state-temporal correlations between streams and initialize a basic prediction model only based on small-scale historical data of the multi-stream. Importantly, this phase requires no pre-defined graphs for correlation graph generation. The self-adaptation process occurs during the online testing phase. We design a continuous graph learning-based structure to handle the concept drift problem. Specifically, we propose an Adaptive Diffusion Graph Attention module (ADGAT) collaborating with the proposed AGG to capture local correlation changes caused by drift and adaptively update the original correlation graph weight. Real-time prediction performance serves as the drift judgment criteria, and different sub-graph updating mechanisms are applied based on drift detection results.

6.3.1 Correlation Graph Structure Initialization

Recent GNN-based research on constructing correlation graphs typically designs their methods based on pre-defined graphs. These pre-defined graphs can provide fixed information, such as sensor locations. While this approach is effective for constructing correlation graph structures, ensuring the availability of pre-defined graphs in real-world applications can be challenging. Additionally, there may not always be sufficient data to train and validate the model in practical situations. To address these two major challenges, we design a novel graph neural network to construct a high-performance spatio-temporal correlation graph structure without pre-defined graphs and initialize the model based on small-scale data.

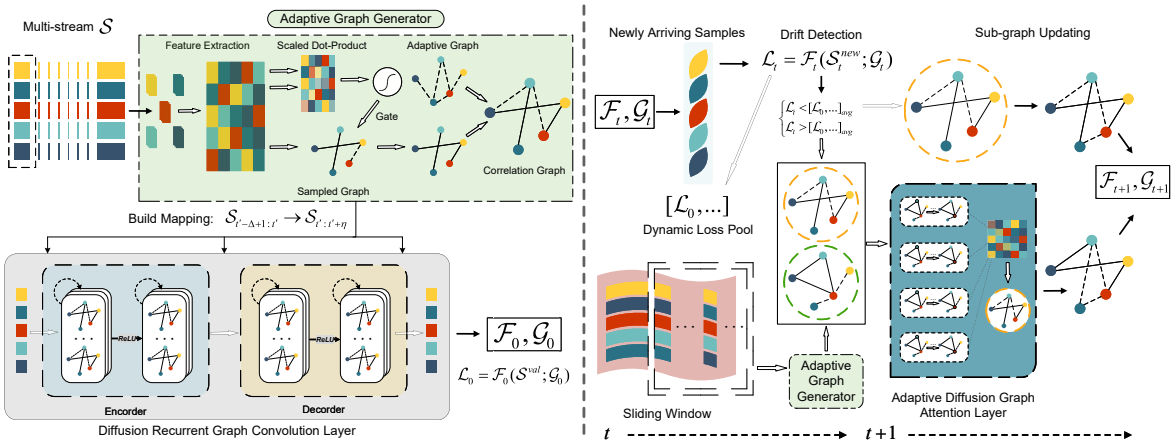


Figure 6.3: Overview of the CGLM framework. The model initialization, depicted on the left, the multi-stream correlation graph structure is learned by the designed Adaptive Graph Generator and embedded in the Diffusion Recurrent Graph Convolutional Layer to map the multi-step prediction tasks. The online testing phase is depicted on the right. Continuous graph learning is achieved by sub-graph updating. As new samples continuously arrive, real-time prediction performance will indicate whether drift has occurred, thereby dynamically applying different sub-graph updating mechanisms.

6.3.1.1 Adaptive Graph Generator

When we do not utilize pre-defined graphs, constructing an effective graph structure based on limited historical information becomes the core task. To achieve this, we design an Adaptive Graph Generator (AGG). The correlation graph structure generated in our proposed CGLM not only simulates the distribution characteristics of historical data but also exhibits good learnability and robustness. This helps address potential concept drift issues that may arise during the online testing process.

Before graph generation, we employ a hyper-network to extract historical data features and compress their dimensions. This entails the application of convolutional layers to the entire training dataset $\mathcal{S}^{tr}$ along the time dimension, followed by vectorization (vec) of the convolutional output (Conv). Then, fully connected layers (FC) are used to reduce the dimensionality further, mapping each stream $\mathcal{S}^u$ in $\mathcal{S}^{tr}$ to a vector z^u. This process is denoted as $z^u = \text{FC}(\text{vec}(\text{Conv}(\mathcal{S}^u)))$. After the feature extraction, each stream is represented as a vector.

To calculate the correlation weight between streams, we first apply the Gumbel-Softmax reparameterization trick [119] to each stream pair u and v. The Gumbel noises $\{\mathrm{G}um_i\}_{i\leqslant N}$ generated from a random uniform distribution are used here to help compute the probability-based correlation weight. Then, we can obtain a sampled graph structure based on all stream pairs, denoted as G_{Gum}:

$$G_{\mathrm{Gum}} = softmax(\frac{\log(\frac{\mu_{\langle u,v\rangle}}{1-\mu_{\langle u,v\rangle}}) + (\mathrm{G}um_u - \mathrm{G}um_v)}{\delta}) \tag{6.2}$$

where $\mu_{\langle u,v\rangle}$ represents the discrete probability variable for the stream pair u and v and δ is the temperature parameter in the Gumbel sampling process.

The sampled graph can be seen as a simulation of the historical data distribution. However, in the presence of concept drift in multi-stream data, it is common for the drift to be localized. In such cases, one or more streams may be affected by drift while the overall correlation remains stable. To address this, we employ a gated attention mechanism on the extracted feature to capture these changes and adjust the model's learning process accordingly. This mechanism assigns greater weight to key streams of the sampled graph, allowing the model to adapt to localized drift effectively.

The calculation process is illustrated in the top left of Fig. 6.3. This mechanism simultaneously treats the extracted feature matrix as the Query and Key vectors. By taking the dot product of these vectors, we obtain the attention scores, which are then passed through a softmax function to derive the attention weights:

$$A_{att} = \mathrm{Softmax}(\frac{z \cdot z^T}{\sqrt{d_z}}) \tag{6.3}$$

where $\sqrt{d_z}$ is the dimension of the extracted feature z.

To enhance the flexibility of the sampled graph, we use the values obtained from applying the sigmoid activation function to the attention weights as the gate: $A_{gate} =$

$Sigmoid(A_{att})$. It controls the weighting process of the attention weights on the sampled graph:

$$G_{WGum} = A_{gate} \cdot A_{att} \cdot G_{\text{Gum}} \tag{6.4}$$

This allows the generation of dynamically weighted sampled graphs during the iterative training process, enhancing its learnability and drift detection capability. Also, this homogenous sampling and weighting design ensures consistency between features and data.

We also integrate an adaptive embedding matrix to improve the robustness of the graph generation. Two dictionaries E_1 and E_2 initialized randomly with learnable parameters present source and target node embedding: $E_1, E_2 \in \mathbb{R}^{N \times d}$, where d denotes the dimension. After initialization, we can obtain the adaptive embedding matrix through the following process:

$$A_{adp} = \text{Softmax}(\text{ReLu}(E_1 E_2^T)) \tag{6.5}$$

The ReLu activation and softmax functions are used to eliminate weak connections between streams and normalize the matrix. In our design, the generated correlation graph structure is embedded into a diffusion convolutional layer, so the applied adaptive matrix can be viewed as the pre-diffusion process. Finally, we obtain the generated self-learned adaptive spatial correlation graph as the graph convolutional learning layer by embedding the designed dynamic weighted sampled graph and adaptive matrix:

$$\mathcal{G} = \sum_{p=0}^{P} G_{\text{WGum}}^{p} \mathcal{S} W_{p1} + A_{adp}^{p} \mathcal{S} W_{p2} \tag{6.6}$$

where P represents the finite steps of graph signal diffusion and W_p represents the p-th order graph convolutional parameters. Through our design, a high-performance

distribution-based graph presenting the deep spatial correlation between streams can be generated. Also, this graph structure can continue to help adapt to concept drift by its learnability in the online testing phase.

6.3.1.2 Temporal Correlation Convolutional Layer

We have a training process similar to traditional GNN-based methods. In CGLM, the adaptive graph generated in the spatial convolutional layer and small-scale historical data are embedded into the recurrent diffusion convolutional layer for further modelling and achieves the prediction mapping task: $S_{t-\Delta+1:t}$ to $S_{t+1:t+\eta}$. This process is performed by a sequence-to-sequence structure to satisfy the flexible multi-step prediction task. The overall hidden state of each time step t^* is updated as follows: h_{t^*-1} to h_{t^*}. The values of multi-stream in the window Δ are used as the input for the encoder, which frequently updates from $t^{'}-\Delta+1$ to $t^{'}$. Then h_{t^*} acting as the representation of $\mathscr{S}_{t^*}$ is used to initialize the decoder, which frequently updates from $t^{'}+1$ to $t^{'}+\eta$.

Considering the directedness of the generated graph structure, we apply the Diffusion Convolutional Gated Recurrent Unit proposed by [86] to achieve the convolutional process, the dual-directional random walk is applied on the generated correlation graph structure $\mathscr{G}$ to achieve diffusion:

$$\Theta_\lambda \star_{\mathscr{G}} Z = \sum_{p=0}^{P} (\theta_{p,1}^{\lambda}(M_O^{-1}\mathscr{G})^p + \theta_{p,2}^{\lambda}(M_I^{-1}\mathscr{G}^T)^p)Z \tag{6.7}$$

where Θ is the filter applied on the diffusion process, M_O and M_I are the out-degree and in-degree matrix. The model parameters for the filters are Θ_λ, where $\lambda = r, q, c$.

$$
\begin{cases}
r_{t^*} = sigmoid(\Theta_r \star_{\mathcal{G}} [\mathcal{S}_{t^*} \parallel \mathrm{h}_{t^*-1}] + b_r), \\
q_{t^*} = \tanh(\Theta_q \star_{\mathcal{G}} [\mathcal{S}_{t^*} \parallel (r_{t^*} \odot \mathrm{h}_{t^*-1}] + b_q), \\
c_{t^*} = sigmoid(\Theta_c \star_{\mathcal{G}} [\mathcal{S}_{t^*} \parallel \mathrm{h}_{t^*-1}] + b_c), \\
\mathrm{h}_{t^*} = c_{t^*} \odot \mathrm{h}_{t^*-1} + (1 - c_{t^*}) \odot q_{t^*}
\end{cases} \tag{6.8}
$$

where r_{t^*} and c_{t^*} are the reset gate and update gate at time step t^*. We use the mean absolute error (MAE) calculation result as the loss function during training. It is calculated between the prediction values $\mathcal{Y} \in \mathbb{R}^{\eta \times I \times J}$ and ground truth $\hat{\mathcal{Y}} \in \mathbb{R}^{\eta \times I \times J}$ in future η time steps:

$$
\mathcal{L} = \mathcal{L}(\hat{\mathcal{Y}}, \mathcal{Y}) = \frac{1}{\eta I J} \sum_{t=1}^{\eta} \sum_{i=1}^{I} \sum_{j=1}^{J} | \hat{\mathcal{Y}}_{ijt} - \mathcal{Y}_{ijt} |, \tag{6.9}
$$

where I represents the number of streams, J represents the dimension of the output. Upon completion of initiated training, we acquire a base model including an initial correlation graph structure for the prediction task: $\{\mathcal{F}_0, \mathcal{G}_0\}$. This base model is tailored to the spatio-temporal correlations present in the historical data and continues to perform its adaptability in the online testing phase, enabling it to adjust to new data distributions.

6.3.2 Online Continuous Graph Learning-based Self-Adaptation

The primary motivation for our self-adaptation method is to address the risks static models face with unknown data distributions. SAGN [132] handles this risk through online learning for concept drift adaptation in multi-streams. However, it overlooks drift severity, lacks targeted graph updates, and creates computational redundancies due to the model pool mechanism. In practical applications, we consider that data streams will be continuously updated over a long period. Therefore, the same update level can erode generalization, causing incorrect updates or overfitting. Meanwhile, simple sub-graph

updating is insufficient for adapting to new distributions, especially with limited initial data. Our proposed CGLM overcomes these challenges, achieving better self-adaptation performance.

Firstly, our proposed novel graph neural networks can learn a spatial-temporal correlation graph structure for multi-stream without pre-defined graphs. Meanwhile, a high-performance basic prediction model can be obtained after the initialization: $\{\mathscr{F}_0, \mathscr{G}_0\}$. Secondly, we employ a drift detection module based on real-time performance and apply different sub-graph updating mechanisms based on the detection results. Thirdly, we design an Adaptive Diffusion Graph Attention Module (ADGAT) module to capture the local correlation changes caused by drift and make sub-graph updating more targeted and accurate. This module works in conjunction with the proposed AGG, dynamically generates new dynamic graphs, and adaptively updates the stream node weight for the original correlation graph. Through this design, we integrate the self-adaptation process with the continuous learning of the correlation graph structure at each time step t: $\{\mathscr{F}_{t-1}, \mathscr{G}_{t-1}\} \rightarrow \{\mathscr{F}_t, \mathscr{G}_t\}$. This process can fully leverage the advantages of graph structures in multi-stream tasks, enabling more precise sub-graph updating while maintaining stable global correlations. This self-adaptation process is illustrated in the right part of Fig. 6.3, and the detailed calculation steps are as follows.

When online testing begins, we initialize a dynamic loss pool B and a sliding window $\mathscr{V}^\tau$ with size τ. We store the loss value $\mathscr{L}_0$ of $\{\mathscr{F}_0, \mathscr{G}_0\}$ on the offline validation set as the first value in B. For each new time step, the model's real-time loss value $\mathscr{L}_t$ is stored in B after obtaining the ground truth, and $\mathscr{V}^\tau$ continuously stores newly arriving samples. If $\mathscr{L}_t$ is less than the mean of the losses stored in B at a certain time step: $\mathscr{L}_t < B_{avg}$, we think the drift does not occur. In this case, the sub-graph updating process relies solely on the weights adjustment applied to newly obtained samples on each stream, and the sub-graph updating rate will be at a lower level. At this point, the self-adaptation

mechanism is expressed as $\{\mathscr{F}_{t-1}, \mathscr{G}_{t-1}\} \xrightarrow{\lambda_1; e^1_{ol}} \{\mathscr{F}_t, \mathscr{G}_t\}$. When $\mathscr{L}_t > B_{avg}$, it indicates that concept drift has occurred. Our CGLM then performs a more targeted and accurate sub-graph updating process, which is completed by two designed modules, AGG and ADGAT.

The AGG used to initialize the correlation graph structure during the offline training phase now is applied to the data within the $\mathcal{V}^\tau$ to generate a new dynamic graph $\mathscr{G}_t^\tau$. Then, our designed ADGAT module performs weight fusion based on this new graph and the original correlation graph to achieve drift self-adaptation. Specifically, ADGAT treats this new graph $\mathscr{G}_t^\tau$ as the real-time dynamic feature of multi-stream at the current time step t. Then, multi-head attention weights are calculated on $\mathscr{G}_t^\tau$. The output of each head is added to a designed diffusion feature vector Λ and embedded into the next iteration:

$$\begin{cases} head_h = Attention(G_t^\tau, G_o, \Lambda), \\ \Lambda += head_h \end{cases} \tag{6.10}$$

Then multiple head outputs are concatenated and projected by the attention weights matrix ω^O to construct a weighted dynamic adjacency matrix G_t^{att} at time step t:

$$G_t^{att} = \text{Concat}(head_1, ..., head_h) \cdot \omega^O \tag{6.11}$$

Meanwhile, we embed an adaptive and learnable parameter σ to adjust the graph updating level:

$$\mathscr{G}_{t-1} = \mathscr{G}_t^{att} = G_t^{att} \cdot \sigma + \mathscr{G}_0 \cdot (1 - \sigma) \tag{6.12}$$

The calculation process for each head is defined as follows. Firstly, we calculate the attention coefficient based on $\mathscr{G}_t^\tau$:

$$\varphi_{uv} = \text{LeakyReLU}(\vec{a}^{T}[\mathcal{W}_u{}^{\tau} || \mathcal{W}_v{}^{\tau}]) \tag{6.13}$$

where $\vec{a}^{T}$ refers an adaptive attention vector, u, v denote stream nodes, $\mathcal{W}^{\tau}$ represents the linear transformation output of $\mathcal{G}_t^{\tau}$. The original graph $\mathcal{G}_0$ is used as the adjacency matrix here and applied to calculate the normalized attention weight between stream nodes, which is denoted as α_{uv}:

$$\alpha_{uv} = \frac{\exp(\varphi_{uv})}{\sum_{k \in \mathcal{N}(u)} \exp(\varphi_{uk})} \tag{6.14}$$

where $\mathcal{N}(u)$ refers to the index of the neighbour nodes of stream node u. Once we obtain α_{uv}, we use it to weight and aggregate the features of neighbouring nodes to update the features of stream node u:

$$\widetilde{\mathcal{W}_u^{\tau}} = \sum_{k \in \mathcal{N}(u)} \alpha_{uk} \mathcal{W}_k^{\tau} \tag{6.15}$$

The obtained attention weights undergo residual connections to avoid the problem of gradient vanishing or exploding during prolonged learning processes. Also, the updated diffusion feature Λ implements diffusion and accumulation across attention head iterations:

$$\widehat{\mathcal{W}_u^{\tau}} = \widetilde{\mathcal{W}_u^{\tau}} + \mathcal{W}_u{}^{\tau} + \Lambda \tag{6.16}$$

In this design, the multi-head attention mechanism is used to capture local correlation changes caused by drift at the current time step, while the diffusion feature mechanism maintains the stability of global correlations. Meanwhile, the learnable parameter σ ensures that the sub-graph updating is adaptively adjusted during long-term continuous

graph learning. We set the base update rate for this situation as e^2_{ol}. However, besides the unpredictability of drift occurrence, the extent of the drift is also unknown. Therefore, a fixed update rate may still lead to underfitting or overfitting during long-term updating. Thus, based on e^2_{ol}, we link the extent of the drift to the updating rate. Specifically, we apply the floor operation to the ratio of $\mathscr{L}_t$ to B_{avg}, and the result is used as the coefficient for e^2_{ol}:

$$\hat{e}^2_{ol} = e^2_{ol} \times \left\lfloor \frac{\mathscr{L}_t}{B_{avg}} \right\rfloor \tag{6.17}$$

At this point, the sub-graph updating rate will be at a dynamic state depending on the extent of drift. This self-adaptation mechanism under drift can be represented as $\{\mathscr{F}_{t-1}, \mathscr{G}_{t-1}\} \xrightarrow{\lambda_2; \hat{e}^2_{ol}} \{\mathscr{F}_t, \mathscr{G}_t\}$. Our designed online continuous graph learning-based self-adaptation method can be summarized as follows:

$$\begin{cases} \{\mathscr{F}_{t-1}, \mathscr{G}_{t-1}\} \xrightarrow{\lambda_1; e^1_{ol}} \{\mathscr{F}_t, \mathscr{G}_t\} & \mathscr{L}_t < B_{avg} \\ \{\mathscr{F}_{t-1}, \mathscr{G}_{t-1}\} \xrightarrow{\lambda_2; \hat{e}^2_{ol}} \{\mathscr{F}_t, \mathscr{G}_t\} & \mathscr{L}_t > B_{avg} \end{cases} \tag{6.18}$$

where λ_1, λ_2 represents different continuous graph learning-based self-adaptation mechanisms applied for non-drift and drift detected situations.

Overall, our designed continual graph learning-based self-adaptation method allows the model to incorporate new local correlation information while maintaining the integrity of the existing correlation graph. Also, Our method enables a more flexible response to the multi-stream concept drift problem while maintaining more robust long-term learning performance.

6.3.3 The Self-adaptation Algorithm of CGLM

We provide the pseudo-code of the self-adaptation process of CGLM, shown in Algorithm 3.

In Step 1 and 2, the initialization and parameter setting for the online testing phase are performed. The base prediction model $\{\mathscr{F}_0, \mathscr{G}_0\}$ obtained from the offline initialization phase, performs the multi-step prediction task for $\mathscr{S}$ starting from Step 3. Once we obtain the ground truth values (Step 4), these newly arriving samples are used to update $\mathscr{V}^\tau$ in Step 5. Meanwhile, the real-time loss value is stored in B (Steps 6 and 7). To detect drift, we compare the loss value $\mathscr{L}_t$ at each time step t with the mean value of values in B (Step 8).

According to the detection results, we apply different mechanisms (λ_1, λ_2) in the sub-graph updating process. If no drift occurs, λ_1 is applied, and sub-graphs are updated only based on a few new samples at a low rate e^1_{ol} in Step 9. When drift occurs, λ_2 is applied. Our designed AGG first generates dynamic feature graphs based on $\mathscr{V}^\tau$ in Step 11. Then, from Step 12 to 13, the proposed ADGAT captures the local correlation changes in the graph by multi-head attention calculation, performs feature diffusion on all heads and updates the original correlation graph weight at σ level. σ is a learnable and adaptive parameter for this updating process. After that, in Step 14, the model is updated at a dynamic rate $\hat{e}^2_{ol}$. Finally, the continual graph learning-based self-adaptation process of this time step $\{\mathscr{F}_{t-1}, \mathscr{G}_{t-1}\} \rightarrow \{\mathscr{F}_t, \mathscr{G}_t\}$ is completed (in Step 9 or Step 14).

6.4 Experiments

6.4.1 Datasets and Baselines

Datasets. We conducted experiments on three large-scale real-world datasets, two from the transportation domain [96, 96, 132] and one from the weather domain [111]. The statistical summary of these three datasets is presented in Table 6.2.

For the weather dataset, the drift is primarily characterized by seasonal variations, along with occasional abrupt changes in weather conditions. Different weather features

Algorithm 3 CGLM continual graph learning-based self-adaptation process

1: Input Multi-stream $\mathscr{S}$, with input historical length Δ, output prediction steps η. The batch size is 1 (New samples arrive one by one). The sub-graph update rates are e^1_{ol}, e^2_{ol} for non-drift and drift detected scenarios, with epoch numbers Q^1_m, Q^2_m. The loss function is $\mathscr{L}$, optimised by Adam [122].

2: Initial The initialized prediction model containing correlation graph structure $\{\mathscr{F}_0, \mathscr{G}_0\ \Theta_0\}$. The dynamic loss pool B is initialized with its first value, $\mathscr{L}_0$, which is calculated by $\{\mathscr{F}_0, \mathscr{G}_0\ \Theta_0\}$ on the offline validation set $\mathscr{S}^{val}$. The sliding window $\mathscr{V}^\tau$ with the window size of τ, is initialized using the historical data of $\mathscr{S}$. For the ADGAT module, the number of attention heads is h, and the adaptive parameter for graph updating is σ. λ_1, λ_2 represents different update mechanisms applied for non-drift and drift detected situations.

for *New samples of* $\mathscr{S}$ *arrive in each time step,* $t = 1, 2, 3, \ldots$ **do**

3: Online Multi-step Prediction:

$\mathscr{F}_t\left(\mathscr{S}^{te}_{t-\Delta+1:t}; \mathscr{G}_{t-1}, \Theta_{t-1}\right) = \mathscr{S}^{pre}_{t+1:t+\eta}$; // Give multi-step prediction results for $\mathscr{S}$

if $t > 1$ **then**

4: Obtain Ground Truth: $\mathscr{S}^{true}_{t:t+\eta-1}$;

5: Update Sliding Window New arriving samples are spliced with the data in $\mathscr{V}^\tau$, while the old data at the far end of the window is discarded;

6: Calculate Current Model Loss:

$\mathscr{L}_t = loss(\mathscr{F}_{t-1}\left(\mathscr{S}^{te}_{t-\Delta:t-1}; \mathscr{G}_{t-1}, \Theta_{t-1}\right), \mathscr{S}^{true}_{t:t+\eta-1})$;

7: Update Dynamic Loss Pool:

$B = [\mathscr{L}_0, \ldots] \leftarrow \mathscr{L}_t$;

8: Drift Detection:

Calculate the mean value of B, $B_{avg} = \frac{1}{n+1} \sum_{n+1}^{n} \mathscr{L}_i$;

if $\mathscr{L}_t < B_{avg}$ **then**

9: Update Sub-graphs under Non-Drift

for $Q = 1, 2, \ldots Q^1_m$ **do** // Sub-graph updating under λ_1, Eq.(6.18)

$\mathscr{G}_{t-1}, \Theta_{t-1} = \text{Adam}(\mathscr{L}_t(\text{Eq.}(7.9)); e^1_{ol})$, $\mathscr{F}_t$;

$\mathscr{G}_t = \mathscr{G}_{t-1}$, $\mathscr{F}_t = \mathscr{F}_{t-1}$;

end

else

10: Update Sub-graphs under Concept Drift

for $Q = 1, 2, \ldots Q^2_m$ **do** // Sub-graph updating under λ_2, Eq.(6.18)

11: Generate Dynamic Graphs: G^τ_t(Eq.(6.7))**;**

12: Calculate Multi-head Attention Weights:

$head_{1\ldots h}, G^{att}_{t-1}$(Eq.(6.10), Eq.(6.11));

13: Update the Original Correlation Graph Weight

$\mathscr{G}_{t-1} = \mathscr{G}^{att}_{t-1} = G^{att}_{t-1} \cdot \sigma + \mathscr{G}_0 \cdot (1 - \sigma)$(Eq.(6.12));

14: Update the Model

$\mathscr{G}_{t-1}, \Theta_{t-1} = \text{Adam}(\mathscr{L}_t(\text{Eq.}(7.9)); \hat{e}^2_{ol}(\text{Eq.}(6.17))$, $\mathscr{F}_t$;

$\mathscr{G}_t = \mathscr{G}_{t-1}$, $\mathscr{F}_t = \mathscr{F}_{t-1}$;

end

end

end

end

exhibit varying degrees of fluctuation and trend shifts during these drift events. For two traffic datasets, the factors causing drift are relatively complex. In addition to periodic drifts associated with time-related patterns such as day-night cycles, there are numerous irregular 'unexpected events' such as accidents or temporary road closures. As shown in Fig. 6.4, we randomly selected one data stream from each of the four multi-streams across the three datasets and extracted a segment to illustrate the trend changes within the selected streams.

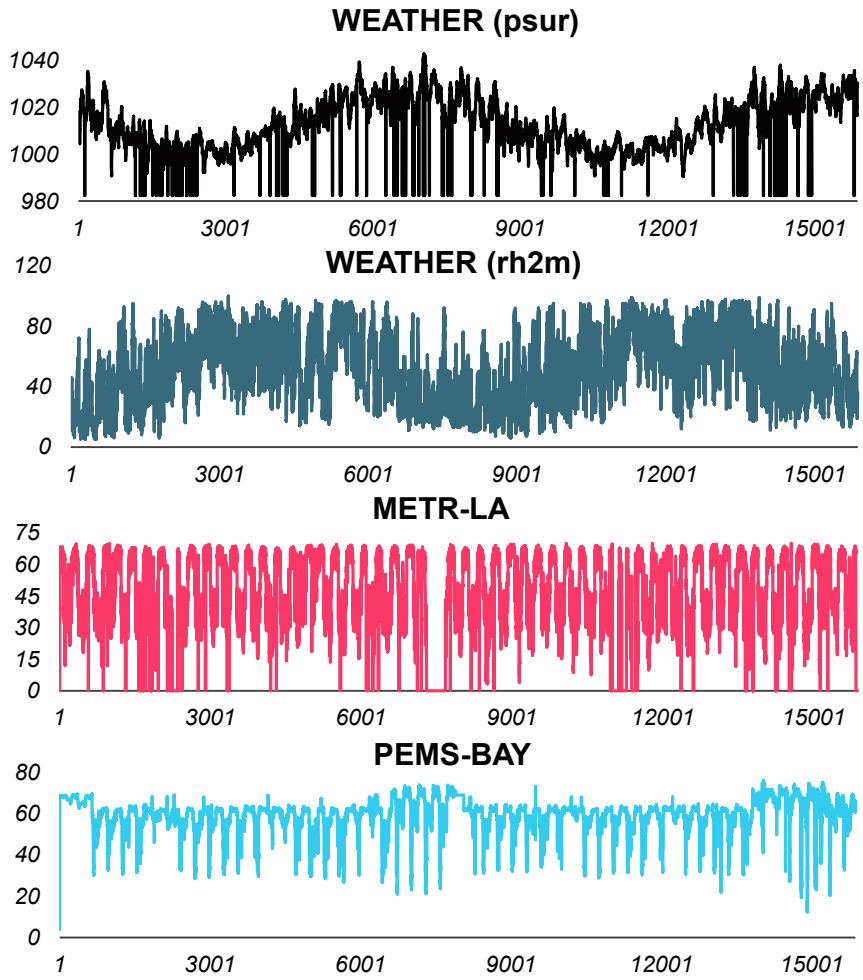


Figure 6.4: Drift patterns in different datasets.

Unlike traditional experimental setups, we will only use small-scale data of the dataset for model training while keeping the testing data unchanged. Since all three multi-

stream datasets experience varying degrees of concept drift over time, this experimental setup aims to assess the model's adaptability to unknown concept drifts along the future timeline. Specifically, in our setup, we allocate the first 20% of the data for training, 10% for validation, and reserve the final 20% for evaluating the model's performance across the two traffic datasets. For the weather dataset, we allocate the first 10% for training, 10% for validation, and reserve the final 40% for evaluating the model's performance.

Table 6.2: Summary of Datasets

DataSet	METR-LA	PEMS-BAY	WEATHER
Stream Nodes	207	325	10
Samples	34272	521160	29639
Update Rate	5min	5min	1h

- **METR-LA** is a publicly available traffic dataset that records the velocity of vehicles on LA County's road network. Loop detectors positioned at 207 places provide the data. In 2012, the dataset spans four months, from March to June [86]. 34,272 time slices overall are produced from the 5-minute intervals at which traffic data is recorded.

- **PEMS-BAY** is a publicly available traffic dataset that gathers traffic data from the California Transportation Agencies' (CalTrans) Performance Measurement System (PeMS) [134]. It includes information from 325 Bay Area sensors and covers a period of six months, from 1 January to 31 May 2017 [86]. 52,116 time slices overall from the 5-minute interval recording of the traffic data.

- **WEATHER** is a publicly available weather dataset gathered from ten weather stations in Beijing, China [125]. It includes hourly weather observations taken between 3:00 am and 3:00 pm, spanning 1188 days from 1 January 2015, to 31

May 2018. All weather stations share a common set of characteristics for weather monitoring. Two particular meteorological parameters, ***psur*** (surface air pressure) and ***rh2m*** (relative humidity at two meters above ground surface), were chosen for our research.

Baselines. We choose a wide range of baseline methods, including conventional machine learning frameworks, standard GNN-based frameworks, and multi-stream self-adaptation frameworks.

Traditional Regression Frameworks:

- **HA**: Historical Average, a method that completes the prediction objective by using a weighted average of historical data;
- **VAR**: Vector Auto-Regression model, a method that predicts each variable based on its own past values and the past values of other variables in the dataset;
- **FNN**: Feedforward Neural Network with two hidden layers and L2 regularization;
- **FC-LSTM**: Fully Connected LSTM, referring to a recurrent neural network architecture where the hidden units of the LSTM modules are fully connected.

GNN-based Methods:

- **DCRNN** [86]: A diffusion convolutional layer is designed to replace the fully connected layer of the GRU [133] in Diffusion Convolutional Recurrent Neural Network to model spatiotemporal dependencies;
- **Graph-WaveNet** [88]: In Graph WaveNet, node embedding is utilized to learn an adaptable dependency matrix in the graph convolutional layer, enabling effective capture of both temporal and spatial dependencies;

- **AGCRN** [95]: Two adaptive blocks are incorporated into the Adaptive Graph Convolutional Recurrent Network to capture dependencies between streams: node adaptive learning and data-adaptive graph;
- **DGCRN** [91]: The Dynamic Graph Convolutional Recurrent Network integrates dynamic graphs, generated by dynamic filters and pre-defined graphs, into a dynamic convolutional recurrent network;
- **ASTGCN** [90]: Attention-based Spatial-Temporal Graph Convolution Networks introduce a novel attention mechanism that operates alongside graph convolution;
- **STFGNN** [89]: Spatial-Temporal Fusion Graph Neural Networks is a data-driven approach that generates temporal maps during training to compensate for deep correlations that spatial maps may not reflect;
- **GTS** [96]: Gumbel sampling is utilized by Graph for Time Series to generate the discrete graph structure among multiple data streams;
- **S^{2}TAT** [135]: The Synchronous Spatiotemporal Graph Transformer (S^{2}TAT) block achieves simultaneous modelling of data by integrating attention mechanisms and graph convolutions.

Multi-stream Self-adaptation Methods:

- **MuNet** [111]: One periodic re-trained predictor and Bayesian connectors are designed to learn and convey new distribution information caused by concept drift. We select ten streams to serve as the traffic datasets for testing because the method is limited to small-scale datasets;
- **SAGN** [132]: A GNN-based concept drift self-adaptation framework, the adaptation process is achieved by sub-graph online learning and periodic validation and reselection of models stored.

- **MSGR** [140]: The graph-regularization-based multi-stream adaptation framework leverages a nearest-neighbor graph structure to construct regularization terms during the online adaptation phase.

6.4.2 Experimental Setup

Platform. The model implementation is done using PyTorch 1.7.1. All experiments are conducted on a server equipped with an Nvidia Quadro RTX 8000 GPU with 48GB of memory and an Intel(R) Xeon(R) Gold 6226R CPU operating at 2.90GHz.

Parameter settings. For the model initialization phase, the temperature parameter δ of the Gumbel-softmax trick is set to 0.9. The training epoch number is 200, and the initial learning rate is 0.005 for all datasets. The diffusion step number is set to 3 for METR-LA and WEATHER ($psur$), 2 for PEMS-BAY and WEATHER ($rh2m$). The sub-graph updating rate e_{ol}^1 under non-drift is set to 0.000005 for all datasets. The base sub-graph updating rate e_{ol}^2 under drift is set to 0.000005 for PEMS-BAY, WEATHER ($psur$) and WEATHER ($rh2m$), 0.00001 for METR-LA. The sub-graph updating epoch number is set to 1 for WEATHER ($psur$) and 3 for other datasets. The sliding window size τ is set to 48. For the ADGAT module, the sub-graph weight fusion parameter σ is set to 0.0001, and the attention head number is set to 4.

Evaluation Metrics. We evaluate the performance of CGLM and all the compared methods using mean absolute error (MAE), mean absolute percentage error (MAPE), and root mean square error (RMSE). First, the predicted values and ground truth values are transformed back to their original scales, and then the evaluation metrics are computed.

Table 6.3: Performance Comparison for SAGN and Baseline Methods on Real-world Datasets

Dataset	Horizon	Metric	HA	VAR	FNN	FC-LSTM	DCRNN	Graph WaveNet	AGCRN	STFGNN	GTS	S2TAT	ASTGCN	DGCRN	MuNet	SAGN	MSGR	***CGLM***
METR-LA	Horizon 3	MAE	9.51	3.98	5.05	3.42	3.70	2.85	3.02	3.43	3.57	3.62	5.04	2.83	8.47	3.68	2.57	**2.49**
		RMSE	13.66	7.35	7.88	6.22	9.82	5.42	5.73	7.69	8.27	7.86	9.89	5.45	8.47	6.49	4.65	**4.46**
		MAPE	30.33%	9.19%	14.51%	9.65%	8.40%	7.54%	8.21%	8.60%	8.16%	8.84%	10.35%	7.42%	17.86%	6.53%	6.80%	**6.42%**
	Horizon 6	MAE	9.51	4.97	5.79	4.03	4.88	3.32	3.46	4.28	4.68	4.37	6.70	3.36	8.50	4.82	2.95	**2.88**
		RMSE	13.66	8.90	8.94	7.31	12.48	6.62	6.87	9.84	10.30	9.69	12.54	6.74	8.50	8.33	5.44	**5.29**
		MAPE	30.33%	11.63%	17.07%	11.67%	10.97%	9.26%	9.88%	11.05%	10.77%	10.92%	13.03%	9.45%	17.91%	8.08%	8.28%	**7.95%**
	Horizon 12	MAE	9.51	6.22	6.73	5.10	6.50	3.85	3.95	5.48	6.15	5.42	8.98	4.35	8.56	6.36	3.39	**3.34**
		RMSE	13.66	10.24	10.18	8.84	15.48	7.81	8.05	12.27	12.48	11.99	15.44	8.58	8.56	10.52	6.27	**6.18**
		MAPE	30.33%	14.66%	20.14%	15.34%	14.43%	11.18%	11.61%	14.63%	14.03%	13.71%	16.62%	12.97%	18.05%	9.88%	9.90%	**9.69%**
PEMS-BAY	Horizon 3	MAE	5.75	1.49	2.18	1.52	1.43	1.37	1.46	1.53	1.41	1.60	1.82	1.38	3.67	1.36	1.34	**1.33**
		RMSE	9.65	2.60	3.98	3.08	3.04	2.89	3.03	3.17	2.76	3.28	3.82	2.88	3.67	2.51	2.48	**2.46**
		MAPE	14.56%	3.00%	4.92%	3.20%	3.05%	2.90%	3.20%	3.35%	3.00%	3.47%	4.16%	2.92%	10.87%	2.88%	2.79%	**2.78%**
	Horizon 6	MAE	5.75	1.90	2.72	2.14	1.85	1.78	1.86	1.95	1.81	1.95	2.34	1.80	3.69	1.72	1.67	**1.65**
		RMSE	9.65	3.39	5.17	4.48	4.21	4.02	4.10	4.31	3.68	4.26	5.05	4.04	3.69	3.27	3.20	**3.16**
		MAPE	14.56%	4.03%	6.47%	4.91%	4.26%	4.10%	4.31%	4.52%	4.14%	4.48%	5.53%	4.12%	10.89%	3.92%	3.75%	**3.70%**
	Horizon 12	MAE	5.75	2.36	3.61	3.04	2.31	2.20	2.24	2.45	2.20	2.35	3.02	2.31	3.75	2.06	1.98	**1.94**
		RMSE	9.65	4.22	6.72	6.12	5.29	4.99	5.02	5.38	4.41	5.12	6.45	5.25	3.75	3.90	3.82	**3.73**
		MAPE	14.56%	5.18%	8.95%	7.47%	5.53%	5.27%	5.34%	5.83%	5.19%	5.53%	7.23%	5.58%	10.99%	4.92%	4.71%	**4.61%**
WEATHER (psur)	Horizon 3	MAE	29.56	1.43	4.85	2.60	1.38	1.37	1.81	2.24	1.21	1.84	2.29	1.15	3.53	1.22	1.30	**1.05**
		RMSE	40.01	4.10	8.01	6.38	4.71	4.92	5.05	5.33	3.04	5.24	5.19	4.59	3.53	1.57	1.62	**1.37**
		MAPE	3.09%	0.15%	0.50%	0.26%	0.14%	0.14%	0.19%	0.23%	0.12%	0.19%	0.24%	0.12%	0.36%	0.12%	0.13%	**0.11%**
	Horizon 6	MAE	29.56	2.23	5.00	3.15	2.12	1.90	2.20	2.69	1.91	2.43	2.81	2.09	3.55	1.89	2.13	**1.65**
		RMSE	40.01	4.77	8.21	7.13	5.74	5.33	5.34	5.65	3.85	5.57	5.62	5.66	3.55	2.32	2.51	**2.01**
		MAPE	3.09%	0.23%	0.51%	0.32%	0.22%	0.19%	0.23%	0.28%	0.19%	0.25%	0.28%	0.21%	0.36%	0.19%	0.22%	**0.18%**
	Horizon 12	MAE	29.56	3.12	5.80	3.79	3.55	2.72	3.05	3.42	3.14	3.18	3.55	4.67	3.66	3.02	3.59	**2.70**
		RMSE	40.01	5.56	8.88	7.69	7.25	5.85	6.00	6.24	5.18	6.08	6.30	8.18	3.66	3.53	4.01	**3.09**
		MAPE	3.09%	0.32%	0.59%	0.39%	0.36%	0.28%	0.31%	0.35%	0.32%	0.33%	0.36%	0.48%	0.37%	0.31%	0.37%	**0.28%**
WEATHER (q2m)	Horizon 3	MAE	20.78	7.28	13.72	10.18	6.37	6.16	7.33	8.49	6.29	8.10	7.53	6.40	18.10	6.80	6.14	**5.63**
		RMSE	24.64	9.99	16.06	13.19	9.21	9.06	10.54	11.76	8.85	11.09	10.35	9.36	18.10	7.91	7.44	**6.94**
		MAPE	49.82%	19.15%	23.65%	17.70%	15.10%	14.74%	15.54%	20.59%	15.68%	19.25%	18.59%	13.77%	57.17%	15.39%	15.31%	**13.69%**
	Horizon 6	MAE	20.78	11.85	18.27	15.50	9.34	8.80	10.77	11.56	9.14	10.92	11.01	10.44	18.19	10.24	8.93	**7.94**
		RMSE	24.64	14.95	20.92	19.06	12.92	12.45	14.68	15.51	12.14	14.68	14.48	14.34	18.19	11.74	10.38	**9.42**
		MAPE	49.82%	32.85%	32.33%	28.52%	23.59%	21.74%	26.84%	29.77%	24.40%	27.56%	28.52%	20.32%	57.45%	22.87%	22.97%	**19.97%**
	Horizon 12	MAE	20.78	16.06	16.55	15.07	12.98	11.52	12.59	12.94	11.89	12.83	12.77	22.15	18.68	13.46	11.62	**10.33**
		RMSE	24.64	19.47	19.41	18.24	17.19	15.83	16.83	17.49	15.27	17.31	16.66	27.72	18.68	15.04	13.14	**11.88**
		MAPE	49.82%	46.85%	28.87%	26.89%	36.79%	29.29%	32.41%	34.76%	33.10%	34.03%	35.27%	27.82%	58.56%	30.95%	30.35%	**26.75%**

Table 6.4: Performance Comparison Based On Large-scale Data Training

	Horizon	Metric	Graph WaveNet	DGCRN	GTS	SAGN	***CGLM***
METR-LA	Horizon 3	MAE	2.69	2.62	2.64	2.46	**2.39**
		RMSE	5.15	5.01	4.95	4.41	**4.28**
		MAPE	6.90%	6.63%	6.80%	6.41%	**6.14%**
	Horizon 6	MAE	3.07	2.99	3.01	2.79	**2.71**
		RMSE	6.22	6.05	5.85	5.15	**5.03**
		MAPE	8.37%	8.02%	8.20%	7.64%	**7.38%**
	Horizon 12	MAE	3.53	3.44	3.41	3.16	**3.09**
		RMSE	7.37	7.19	6.74	5.91	**5.82**
		MAPE	10.10%	9.73%	9.90%	9.10%	**8.85%**

6.4.3 Experimental Results

We evaluate CGLM on three real-world datasets of varying scales. They have different numbers of stream nodes; the two traffic datasets, METR-LA and PESM-BAY, have more stream nodes, while the weather dataset has fewer stream nodes. This also demonstrates the adaptability of our proposed CGLM to data streams of varying scales.

Table 6.3 presents the overall comparison results. Our method outperforms other baselines in multi-step prediction tasks across all datasets when only small-scale data is available for model initialization. For the four multi-streams across the three datasets, our CGLM improves predictive performance over the state-of-the-art GNN-based model by 2.46%, 1.41%, 12.75%, and 9.57%, respectively. The METR-LA and WEATHER ($rh2m$) datasets contain more drifts, whereas the PEMS-BAY and WEATHER ($psur$) datasets exhibit more regular and consistent data distributions. This indicates that our CGLM is not limited by the degree or frequency of drift occurrences. The continual graph learning-based design allows the model to dynamically adjust the self-adaptation mechanism and level based on the drift's extent.

Additionally, we conduct further comparison tests using several high-performance GNN-based methods on the METR-LA dataset, as shown in Table 6.4. In these tests, the first 70% data of the dataset is used for initial model training, 10% data is used for the validation, and the test data remains unchanged. It can be observed that traditional static models show improved prediction performance due to covering more data distributions during training. However, our proposed CGLM still maintains the best prediction performance.

By comparing Table 6.3 and Table 6.4, it can be seen that when the amount of data used for initial training is significantly reduced, concept drift has a noticeable impact on the performance of static models. Most models exhibit a decline in performance due to the inability of the training data to cover the unknown data distributions in the future timeline. Even though there are relatively regular changes in these datasets, maintaining high-performance predictions continuously requires the model's adaptive capabilities.

To demonstrate the practical applicability of the proposed CGLM framework in real-world scenarios, we record the runtime and compare it with baseline methods. For fairness, all methods are initialized using 70% of the METR-LA dataset during offline training. As shown in Table 6.5, our method exhibits the longest computation time; however, the increase is not exponential, and remains within a reasonable range.

More importantly, as listed in the dataset descriptions, the fastest sample update frequency in our benchmark is found in the two traffic datasets, where new data arrives every 5 minutes. In contrast, the average time required for online adaptive learning at each time step in our framework is significantly shorter than this interval. This means our model can complete its online adaptation well within the data update window, ensuring that there is no delay in prediction or downstream application.

Table 6.5: Runtime Comparison on METR-LA Dataset

	Computational Stage	
Method	Offline Training (s/epoch)	Online Testing (s/sample)
GTS	48.87	0.0006
SAGN	49.24	0.34
MSGR	51.96	1.06
SAGN	53.34	1.15

Table 6.6: Ablation Study

	Horizon	Metric	Graph WaveNet	w/o ol	w/o agg	w/o adp	***CGLM***
METR-LA	Horizon 3	MAE	2.85	2.74	3.24	2.60	**2.49**
		RMSE	5.42	5.18	5.61	4.43	**4.46**
		MAPE	7.54%	7.05%	9.04%	6.90%	**6.42%**
	Horizon 6	MAE	3.32	3.22	3.92	3.07	**2.88**
		RMSE	6.62	6.27	6.98	5.69	**5.29**
		MAPE	9.26%	8.87%	11.55%	8.69%	**7.95%**
	Horizon 12	MAE	3.85	3.76	4.91	3.63	**3.34**
		RMSE	7.81	7.39	8.54	6.73	**6.18**
		MAPE	11.18%	11.08%	15.13%	10.65%	**9.69%**

6.4.4 Ablation Study

We perform ablation experiments on the METR-LA dataset to further validate the effectiveness of the various components in our framework. Graph-WaveNet is the GNN-based model with the best performance initialized on small-scale data in Table 6.3, we also add it to the comparison to show the performance difference.

w/o ol refers to the model operating entirely in offline mode, meaning that when new samples arrive, the model does not receive any updates and relies solely on the initialized model to perform the prediction task on unknown data. From Table 6.6, it can be seen that even in offline mode, our method maintains the best performance. It also shows the adaptive correlation graph structure generated by our designed AGG has better

generalization performance than pre-defined graphs.

w/o agg denotes that we remove the attention-weighted part of the sampled graph during the graph generation process in the AGG module. We only use the sampled graph to construct a multi-stream correlation graph structure. Additionally, there is no drift detection or ADGAT module during the online testing phase. Sub-graph updating relies solely on fine-tuning the node weights with new samples. As shown in Table 6.6, its performance declines when the model lacks adaptive modules. Although new samples can still help adjust the sub-graph weights, this undifferentiated, lightweight update method is insufficient for adapting to unknown concept drift.

w/o adp represents that the ADGAT module is removed. The initialized model still includes the complete AGG module. However, during the online testing phase, there will no longer be a sliding window to store new samples for dynamic graph generation and weighting fusion between the original and new graphs. Instead, only lightweight sub-graph updating will be performed using the new samples. It can be seen that, compared to the results of **w/o ol**, the model's performance improves due to the adjustment of sub-graph weights with new samples. However, the self-adaptation mechanism must still be active during the online testing phase to better adapt to concept drift.

To provide a clearer understanding of our proposed method's performance during the online adaptation phase, we present a comparison of loss trends between our method and existing online adaptation approaches when drift occurs in multi-stream METR-LA (the average loss of these three methods on the validation set are all below 2.75). As shown in Fig. 6.5, our method demonstrates enhanced adaptability to new data distributions and achieves superior average prediction performance compared to the two existing approaches.

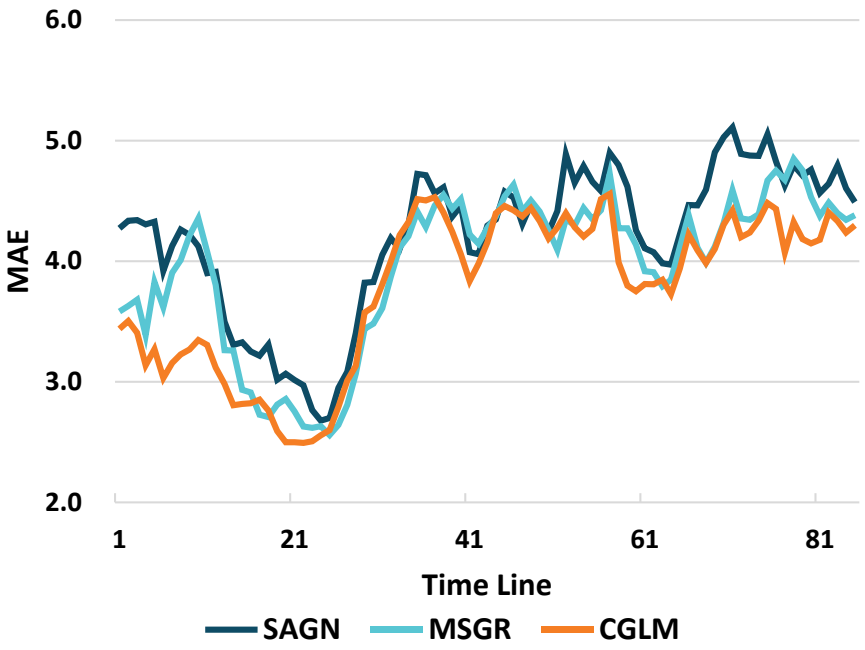


Figure 6.5: Online Drift Adaptation Comparison.

6.4.5 Parameters Sensitivity

The parameter sensitivity experiments are constructed in this section. Four parameters in our framework are tested: the sliding window size τ for dynamic graph generation, the sub-graph update rate e_{ol}^1, e_{ol}^2 for non-drift and drift scenarios, the adaptive parameter σ for weight fusion during original correlation graph updating. MAE is used as the metric in this section, and all testing results are shown in Fig. 6.6.

The parameter τ is the size of the sliding window, and it determines how many recently obtained new samples are used to generate the new dynamic adaptive graphs. Through testing (Fig. 6.6 (a)), it can be seen that this parameter is not sensitive. This indicates that our designed ADGAT can effectively capture changes in the latest samples. However, considering the efficiency of online computation, this parameter should not be set too large to avoid unnecessary computational burden.

The sub-graph update rate e_{ol}^1 and e_{ol}^2 (Fig. 6.6 (b), (c)), represents different learning degrees under non-drift and drift situations. It can be observed that both e_{ol}^1 and e_{ol}^2

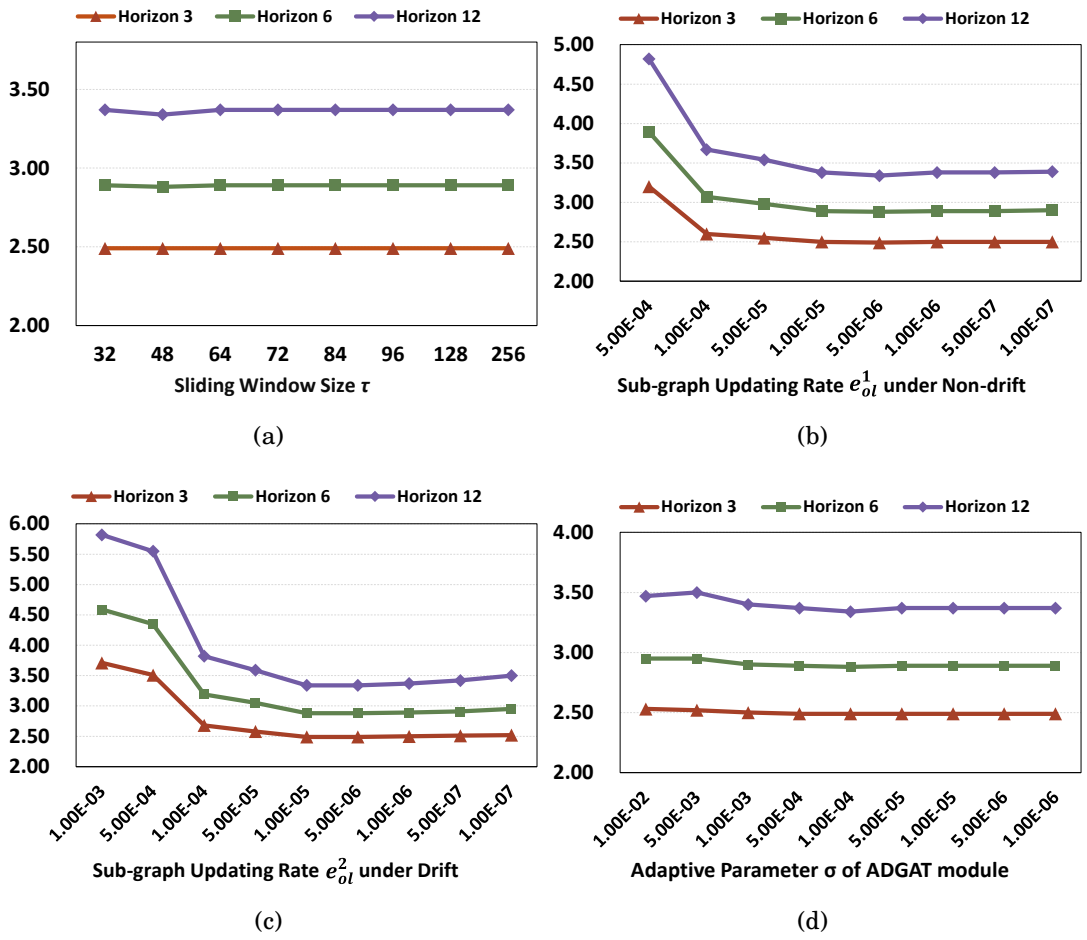


Figure 6.6: The effect of different parameters in the online self-adaptation phase.

should not be set too large, as this may lead to erroneous updates or overfitting. In our framework, the role of e^1_{ol} is to maintain model performance stability by updating sub-graphs lightly. e^2_{ol} is a base updating rate for drift-detected scenarios. The model automatically adjusts e^2_{ol} based on the extent of drift according to Eq.(6.17).

The σ represents the initial coefficient for updating the weights of the original graph structure in ADGAT. This coefficient will continuously adapt and adjust during the online testing phase, but the initial value should not be set too large. It can be observed from Fig. 6.6 (d) that an excessively large initial value may cause the model's weights to concentrate too heavily on drifting regions, leading to a risk of overfitting.

6.5 Summary

In this paper, we propose a continuous graph learning-based multi-stream concept drift adaptation method, CGLM. This framework is designed around a multi-stream correlation graph structure and achieves self-adaptation to new data distributions through adaptively driven sub-graph updating. Unlike traditional GNN model training methods that rely on pre-defined graphs and large amounts of historical data, our approach introduces a new graph neural network architecture embedded with an adaptive graph generation module, AGG. This allows the construction of a highly generalizable multi-stream correlation graph structure and initialising the base prediction model using only small-scale training data.

When multi-stream updates, drift detection results trigger different continuous graph learning-based self-adaptation mechanisms to complete the sub-graph updating process. The lightweight adjustment for sub-graphs is applied under non-drift conditions. Correspondingly, we design the ADGAT module, working with the proposed AGGto handle drift, which uses multi-head attention to capture drifting nodes within sub-graphs and maintains global stability through a diffusion mechanism. The adaptive parameter σ automatically adjusts the degree of weight fusion. Under this adaptive mechanism, any distribution and local correlation changes in the multi-stream data are promptly captured and addressed, enabling the model to continuously provide high-performance multi-step predictions across all streams. Extensive and comprehensive experiments demonstrate the effectiveness and superior performance of CGLM.

CHAPTER 7

MULTI-SCALE ADAPTIVE CONVOLUTIONAL GRAPH FOR MULTI-STREAM CONCEPT DRIFT

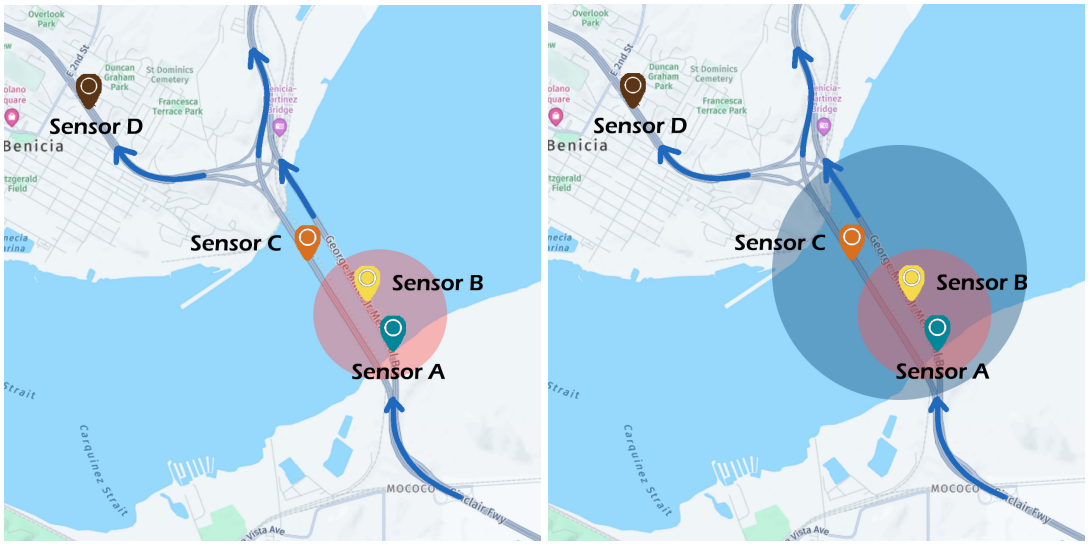
7.1 Introduction

In streaming environments, data distributions often change over time, a phenomenon known as concept drift [53]. This occurs frequently in real-world scenarios, such as shifting topics in social media discussions or abrupt changes in weather and traffic patterns [141]. Concept drift is inherently unpredictable. Moreover, the rapid growth of data sources and the speed of information generation adds further complexity to prediction tasks in such contexts [56]. Traditional machine learning methods typically assume consistency between training and test data distributions [99]. However, when static models fail to adapt to new distribution patterns in dynamic data streams, their prediction performance can degrade significantly [107]. This limitation has led to the development of concept drift adaptation techniques aimed at improving real-time prediction in the face of changing data distributions [136].

The goal of concept drift adaptation is to enable models to recognise and learn new distribution patterns from incoming data, thereby addressing the concept drift problem [142]. Adaptation mechanisms are typically activated by pre-defined rules or drift detection modules [137]. While these approaches perform well in single-stream scenarios, they also exhibit certain limitations [114]. In a single-stream context, the focus is primarily on isolated drift patterns and their effects. However, in multi-stream environments, there may be spatial and temporal correlations between streams, meaning that drift does not always occur in isolation. Instead, multiple streams may be impacted simultaneously [130]. In such situations, constructing separate adaptation frameworks for each stream becomes inefficient and difficult to scale. As a result, developing a concept drift adaptation framework based on multi-stream correlations has emerged as a significant challenge in the field [111].

The correlation among multiple streams is essential for constructing an effective concept drift adaptation framework [138]. Beyond reducing computational redundancy by avoiding repeated use of single-stream adaptation frameworks, this correlation introduces more complex drift patterns. When drift occurs—such as congestion, accidents, or road maintenance—its impact may spread synchronously or asynchronously to multiple connected routes [140]. Figure 7.1 illustrates an example of multi-stream correlation changes. There are two cross-sea passages in the figure. Due to their proximity, congestion on the road segment monitored by sensor A leads to simultaneous congestion on the segment monitored by sensor B (concurrent drift). Soon after, the increased volume of vehicles entering another cross-sea passage causes congestion on the road monitored by sensor C, which is affected by the previous congestion (delayed drift). Furthermore, sensor D, being located farther away, may not be affected if congestion on the cross-sea passage is quickly resolved. In real-world applications, various complex correlation changes can also occur due to temporary maintenance, road repairs, and other factors

unrelated to the distance between locations.



(a) Concurrent Drift (b) Delayed Drift

Figure 7.1: In traffic scenarios, multi-stream correlations can change dynamically. Sensors generate data streams, and when drift occurs, different streams may experience its impact on different streams may be concurrent or delayed, occurring at varying time steps.

In recent years, research on multi-stream adaptation frameworks has gradually gained attention. However, existing frameworks still exhibit certain limitations. Some methods are designed specifically for two to three data streams, requiring one stream to serve as the source and the others as targets [143], akin to the source and target domains in transfer learning [144]. Additionally, some approaches are tailored for small-scale multi-stream scenarios and specific drift types [111]. Some studies have explored using graph neural networks to capture and represent the correlation among multiple sequences, thereby driving prediction tasks [139]. Motivated by these studies, [132] and [140] introduce GNN-based multi-stream adaptation frameworks with similar motivations, but the former only relies on simple online learning, while the latter focuses more on the graph structure constraints of adaptation learning.

To further bridge gaps, we propose a novel multi-scale adaptive convolutional graph

framework, MACG, to address multi-stream concept drift by tackling two key challenges: constructing a generalizable multi-stream correlation graph during initial training and enabling proactive model updates to predict and adapt to drift during online testing. To achieve this, we design adaptive graph convolutional layers that apply multi-scale convolutions across temporal scales to historical data, followed by independent multi-sampling. This captures complex spatio-temporal correlations and produces a highly generalizable graph structure. During online testing, the model adapts dynamically based on real-time feedback. In non-drift periods, lightweight sub-graph updates are performed using new samples. When drift is detected, a more robust strategy is applied. Firstly, the multi-scale adaptive convolutional layers used in training are performed on a sliding feature window to generate a new graph, which is fused with the previous graph by a learnable parameter using an exponentially weighted moving average (EWMA) style. The proactive drift prediction module matches current drift patterns with historical ones to guide updates. To prevent mismatches with unknown drift patterns, we incorporate a shadow model that performs synchronized updates using a data sliding window. Dynamically random validation ensures that model adjustments proceed in the correct direction.
The contributions of this chapter are as follows:

- We introduce MACG, a multi-scale adaptive convolutional graph framework designed to overcome concept drift in multi-stream. MACG performs multi-sampling on historical data across different time-scale convolutions to initialize a highly generalizable correlation graph structure without pre-defined graphs during training. This approach captures various levels of spatial-temporal dependencies between streams, ensuring that the generated graph structure provides a more flexible and comprehensive representation of data characteristics.

- The novel adaptation learning module, is designed to facilitate model updates for effective drift adaptation. When drift is detected, MACG first generates a new

correlation graph by applying the same multi-scale adaptive convolutions used during training to the feature sliding window, then fuses it with the previous graph. Next, MACG leverages historical data to match potential drift patterns and combines them with new samples and the fused graph to predict the current drift and update sub-graph weights. Furthermore, a shadow model synchronously updates using only new samples, while dynamic random validation mitigates the risk of erroneous updates caused by misinterpreted drift trends.

- In this chapter, extensive experiments are conducted on three large-scale real-world datasets. The results indicate that MACG achieves significant improvements over state-of-the-art baseline methods.

The work described in this chapter is based on the manuscript "Multi-Scale Adaptive Convolutional Graph for Multi-Stream Concept Drift" under consideration by IEEE-TCYB.

7.2 Problem Statement

Before introducing our proposed framework, we define our problem first in this section. We summarise important notations of our settings in Table 7.1.

Definition 7.1 (Multi-Stream)**.** A multi-stream, represented as $\mathscr{S}$, consists of multiple data streams $\mathscr{D}^i$, where $\mathscr{S} = \{\mathscr{D}^1, \mathscr{D}^2, \ldots, \mathscr{D}^i\}$. Each individual data stream $\mathscr{D}^i$ within $\mathscr{S}$ includes a set of feature time series, defined as $\mathscr{D}^i = \{\mathscr{T}^1, \mathscr{T}^2, \ldots, \mathscr{T}^j\}$. At a given time step t, the value of a particular time series $\mathscr{T}^j$ is represented by X_t^j.

Given the complexity of multi-stream environments, this chapter assumes that each data stream $\mathscr{D}^i$ in the multi-stream $\mathscr{S}$ contains identical feature time series. For instance, in an urban traffic network, monitoring points represent a multi-stream, where each

Table 7.1: Summary of Notations

Notations	Description
$\mathscr{S} = \{\mathscr{D}^1, \cdots, \mathscr{D}^i\}$	A multi-stream contains i data streams.
$\mathscr{D} = \{\mathscr{T}^1, \cdots, \mathscr{T}^j\}$	A data stream contains j time series.
$\mathscr{T}^j = \{\mathrm{X}_t^j\}$	A time series in the data stream.
X^j	The feature value of the time series.
$t \in \mathbb{Z}^+$	The time step.
$\mathscr{P}$	The probability distribution.
Δ	The past window size.
η	The prediction time steps.
$\mathscr{G}$	The correlation graph structure of multi-stream.
$\mathscr{F}$	The predictor for multi-stream.
$\mathscr{S}^{tr}, \mathscr{S}^{val}, \mathscr{S}^{te}$	The training, validation and testing sets.

stream consists of multiple feature time series such as speed or vehicle count. Our study focuses on shared features across these streams, where identical feature series from different data streams collectively form the multi-stream in the task at hand.

Definition 7.2 (The Correlation in Multi-Stream)**.** The spatio-temporal correlation between streams in the multi-stream $\mathscr{S}$ is defined as $\mathscr{G}$. $\mathscr{G}$ also includes the potential correlations between drifts caused by complex spatio-temporal dependencies.

When discussing spatio-temporal correlation between streams, both fixed and dynamic factors should be considered. For instance, in urban traffic monitoring networks, sensor location information typically serves as a fixed factor. However, traffic flow between roads can be mutually influenced by dynamic factors like traffic accidents, resulting in complex drift scenarios and relationships. These dynamics include a higher-dimensional spatio-temporal correlation change between streams. For a multi-stream, the global

correlation typically remains stable, while the local correlation may change over time caused by concept drift.

Definition 7.3 (Concept Drift in Multi-Stream)**.** The data distribution is represented as $\mathscr{P}$. Concept drift occurs at time step $t+1$ if the data distribution at that time, $\mathscr{P}_{t+1}(\mathscr{S})$, differs from the distribution at the previous time step, $\mathscr{P}_t(\mathscr{S})$. The concept drift in the multi-stream is identified when drift occurs in any stream within $\mathscr{S}$.

Definition 7.4 (Concept Drift Correlation in Multi-stream)**.** In multi-stream environments, drift correlations can be categorized based on the time difference between drifts. Concurrent drift refers to concept drift occurring simultaneously across multiple streams. On the other hand, delayed drift occurs when there is a time gap between the drifts in different streams. A detailed definition of the correlation between drifts in multi-stream settings is provided in [132].

Problem 5 (Adaptation Learning for Multi-Stream at t-step with Dynamic Correlation Graph $\mathscr{G}_t$)**.** *At time step t, given the observed δ-step data and the correlation structure $\mathscr{G}_(t-1)$, to map next η-step data and $\mathscr{G}_t$, formalized as follows:*

$$\mathscr{F}_t, \mathscr{G}_t = \underset{f \in \mathscr{F}, \mathscr{G}}{\arg\min}\ \ell\left(f\left(\mathscr{S}_{t-\Delta+1:t}, \mathscr{G}_{t-1}\right), \mathscr{S}_{t+1:t+\eta}\right). \tag{7.1}$$

where $\mathscr{F}_t, \mathscr{G}_t$ are learned adaptively from $\mathscr{F}_{t-1}, \mathscr{G}_{t-1}$.

Our task shares some similarities with traditional multi-sequence prediction tasks, but differs in that we consider the concept drift problem within the test set. These drifts may negatively impact the model's prediction performance. In our setting, the initial multi-stream correlation graph structure is learnable and adaptive, and the test phase is conducted online. The model is no longer static; instead, it undergoes adaptation learning with continuously incoming new samples.

7.3 Methodology

In this section, we provide a detailed introduction to the proposed MACG framework. Figure 7.2 presents an overview of the framework's design. We address model adaptation to concept drift from both the initialization and online testing phases. During initialization, we design a novel adaptive graph convolution module to help generate a multi-stream correlation graph structure with high generalizability. Specifically, we apply convolutions at multiple temporal scales on historical data and perform multi-sampling to capture complex spatio-temporal correlations between streams, especially for recognizing dynamic shifts at different frequencies across streams.

During the online testing phase, real-time prediction performance serves as an indicator for concept drift detection. Based on the detection results, tailored adaptation strategies are employed to ensure accurate sub-graph updates. In non-drift scenarios, new samples directly guide sub-graph updates. When drift is detected, multi-scale adaptive convolutions are applied to a feature sliding window to generate a new correlation graph, which is then fused with the previous graph. A proactive drift forecasting module further matches and predicts potential drift patterns using historical data. To prevent erroneous updates, a shadow model and a dynamic random validation mechanism are incorporated into the framework.

7.3.1 Spatio-temporal Correlation Graph Initialization

In recent years, GNN-based frameworks commonly rely on pre-defined graphs, often incorporating location information from multi-stream sensors, to construct spatio-temporal correlation graphs. However, this approach faces challenges such as the unavailability of pre-defined graphs, reliance on fixed location information, and limited generalizability needed for adapting to concept drift. To overcome these issues, we propose a novel spatio-temporal correlation graph construction method based entirely on historical data.

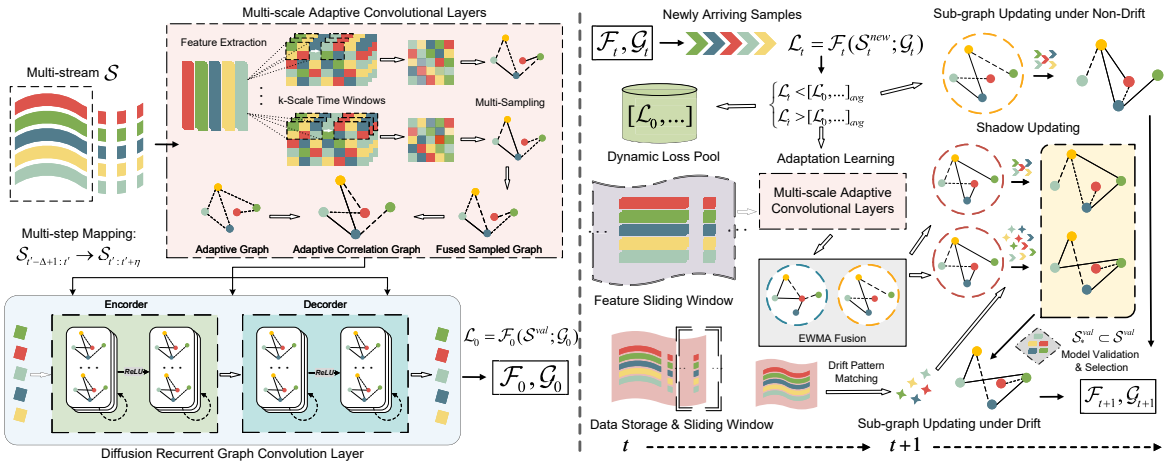


Figure 7.2: Overview of the MACG framework. The left portion depicts the model initialization phase, utilizing historical data to construct a highly generalized multi-stream correlation graph structure. Multi-scale adaptive convolutional layers are designed to individually extract features across diverse time scales, incorporating multi-sampling techniques. These features are iteratively integrated into a diffusion recurrent graph convolution layer, enabling the model to capture intricate spatio-temporal correlations. The right portion illustrates the online testing phase, where new samples come continuously. Real-time prediction performance serves as a drift detection indicator, guiding the model's adaptation through tailored strategies.

7.3.1.1 Multi-Scale Adaptive Correlation Graph

Although drift occurrence is difficult to predict and may involve unknown data distributions, thoroughly leveraging historical data remains essential when constructing models. Historical data contains complex spatio-temporal correlations between streams, particularly with reference to time, where non-synchronous variation patterns may exist across different streams. These characteristics play a significant role in enhancing model generalization to adapt to concept drift.

To address this, we design a multi-scale adaptive convolutional graph structure to fully explore the complex relational patterns in historical data. Existing work, such as [96], also recognized the limitations of pre-defined graphs and used a sampling-based approach to construct adjacency-matrix-like structures. Although this method has some limitations in capturing complex spatio-temporal information, it provided

us with valuable insights. In our design, we integrate multiple convolutional layers of different time scales for feature extraction. These convolutional layers are embedded into independent hypernetworks that capture both synchronous and asynchronous patterns across streams within multiple time windows along the temporal dimension. Specifically, we treat all the historical data from the multi-stream $\mathscr{S}^{tr}$ as a whole, and for each different time scale, we perform two convolution sets:

$$\begin{cases} \text{Conv}_1^k : H_1^k = \text{ReLu}(\text{BN}_1(\text{Conv}_1^k(\mathscr{S}^{tr}))) \\ \text{Conv}_2^k : H_2^k = \text{ReLu}(\text{BN}_2(\text{Conv}_2^k(H_1^k))) \end{cases} \tag{7.2}$$

where k represents different time scales, We can adjust k based on the complexity of the multi-stream data and the prediction requirements. In this chapter, based on our multi-step prediction task, k is set to 2, which allows for flexible capturing of dependencies between streams at both long and short time steps. Then, the fully connected layers (FC) are used to achieve vectorization: $z^k = \text{FC}(\text{vec}(H_2^k))$. Each stream is mapped into one dimension in z^k.

Then, to calculate the probability-based correlation weights between streams and generate the sampled graph, we first use a one-hot, non-diagonal interaction graph that treats each stream node as both a "receiver" and "sender." This setup allows us to simulate directed relationships between stream nodes via feature combinations, represented as an N-dimensional vector, Ω. Next, the Gumbel-Softmax reparameterization trick [119] is applied to each stream node pair $\langle \mathscr{D}^u, \mathscr{D}^v \rangle$ in the vector Ω, resulting in a sampled graph structure, denoted as G_{Gum}:

$$\begin{cases} \{\text{G}um_i\}_{i \leqslant N} = -\log(-\log(U_i)), U_i \sim U(0,1) \\ G_{\text{Gum}} = \text{softmax}(\log(\dfrac{\mu_{\langle u,v \rangle}}{1-\mu_{\langle u,v \rangle}}) + (\text{G}um_u - \text{G}um_v))/\delta \end{cases} \tag{7.3}$$

where $\{\mathrm{G}um_i\}_{i\leqslant N}$ are independently sampled Gumbel noise terms, adding randomness to the logits of each potential pairwise connection and introducing variability that captures the stochastic nature of real-world connections. These terms support differentiable sampling, allowing gradients to flow during training. Here, $\mu_{\langle u,v\rangle}$ represents the discrete probability of the connection between streams $\mathscr{D}^u$ and $\mathscr{D}^v$, while δ is the temperature parameter that controls the smoothness of the sampling distribution.

While the graph generated through Gumbel sampling captures the overall distribution of historical data, concept drift in multi-streams often occurs locally. It usually affects specific subsets of streams or time periods. To capture these patterns, we perform multiple independent Gumbel samplings on k extracted multi-stream feature vectors $\{\Omega_i\}_{i\leqslant k}$. These feature vectors are processed using k-scale convolutions. The multi-scale convolution captures asynchronous changes across streams at different temporal granularities. This is crucial for identifying local and temporal variations. By averaging k independently sampled graphs, as shown in Eq. (7.4), we reduce noise and stabilize the model. This approach ensures robustness against spurious fluctuations while maintaining adaptability to meaningful drift:

$$G_{\mathrm{AGum}} = \frac{1}{k}\sum_{\mathrm{i=1}}^{\mathrm{k}} G^i_{\mathrm{Gum}} \tag{7.4}$$

To enhance the robustness and learnability of the graph generation process, we introduce an adaptive embedding matrix. Two dictionaries, E_1 and E_2, are initialized with learnable parameters to represent the source and target node embeddings, respectively: $E_1, E_2 \in \mathbb{R}^{N\times d}$, where d is the embedding dimension. The adaptive embedding matrix A_{adp} is computed by taking the dot product of E_1 and E_2, followed by a ReLU activation and a softmax function:

$$A_{adp} = \mathrm{Softmax}(\mathrm{ReLu}(E_1 E_2^T)) \tag{7.5}$$

The ReLU activation function removes weak connections between streams, while the softmax function normalizes the matrix, ensuring weights represent the relative importance of node interactions. This adaptive matrix acts as an additional layer to model dynamic relationships between stream nodes, capturing the evolving correlations between senders and receivers in sampled graphs.

In our design, the generated correlation graph structure is then embedded into a diffusion convolutional layer, where the self-learned adaptive spatial correlation matrix plays a crucial role. It serves as a pre-diffusion process, refining the learned graph structure before applying graph convolutions to better capture the correlations between streams. Through this approach, the adaptive matrix dynamically adjusts the pairwise correlations, allowing the model to better represent the actual data streams and respond to changing patterns in the data.

Finally, the dynamic weighted sampled graph and the adaptive matrix are fused and embedded into the diffusion process as the multi-stream correlation graph structure:

$$\mathscr{G} = \sum_{p=0}^{P} G^{p}_{\mathrm{AGum}} \mathscr{S} W_{p1} + A^{p}_{adp} \mathscr{S} W_{p2} \tag{7.6}$$

where P represents the finite steps of graph signal diffusion and W_p denotes the p-th order graph convolutional parameters. This design enables the generation of a high-generalizability, distribution-based graph that captures deep spatio-temporal correlations between streams. Moreover, this graph structure is flexible and can continue to adapt to concept drift during the online testing phase, enhancing the model's ability to handle dynamic data streams effectively.

7.3.1.2 Adaptive Graph Convolutional Recurrent Layer

In MACG, we employ a sequence-to-sequence model with an encoder-decoder structure to achieve the multi-step mapping tasks for multi-stream: $S_{t-\Delta+1:t}$ to $S_{t+1:t+\eta}$. The

multi-scale adaptive graph initialized by the spatial convolution layer is embedded into the recurrent diffusion convolution layer to update hidden state at each time step t^*. Specifically, the multi-stream values within the window Δ are used as inputs for the encoder, which is updated hidden state iteratively from $t^{'} - \Delta + 1$ to $t^{'}$: h_{t^*-1} to h_{t^*}. The updated state h_{t^*}, representing $\mathscr{S}_{t^*}$, is then used to drive the recurrent update of the decoder, from $t^{'} + 1$ to $t^{'} + \eta$.

Our multi-scale adaptive correlation graph is designed to capture spatial-temporal relationships, including the directionality between stream nodes. Consequently, we apply the Diffusion Convolutional Gated Recurrent Unit (DCGRU), as proposed by [86], which is well-suited for directed graphs, to perform convolutional computation. Also, after the adaptive matrix completes the preliminary diffusion for the initialized correlation graph structure, a bidirectional random walk is used to achieve the diffusion process to further enhance the robustness of the $\mathscr{G}$:

$$\Theta_\lambda \star_{\mathscr{G}} Z = \sum_{p=0}^{P} (\theta_{p,1}^{\lambda} (M_O^{-1}\mathscr{G})^p + \theta_{p,2}^{\lambda} (M_I^{-1}\mathscr{G}^T)^p) Z \tag{7.7}$$

where Θ represents the filter applied to the diffusion process, with out-degree and in-degree matrices M_O and M_I for forward and reverse propagation. The parameters Θ_λ are learned filters, where λ denotes specific values r, c, and q for reset, update, and candidate gates.

The recurrent convolutional computation using these diffusion-enhanced gates proceeds as follows:

$$\begin{cases} r_{t^*} = sigmoid(\Theta_r \star_{\mathscr{G}} [\mathscr{S}_{t^*} \parallel \mathrm{h}_{t^*-1}] + b_r), \\ c_{t^*} = sigmoid(\Theta_c \star_{\mathscr{G}} [\mathscr{S}_{t^*} \parallel \mathrm{h}_{t^*-1}] + b_c), \\ q_{t^*} = \tanh(\Theta_q \star_{\mathscr{G}} [\mathscr{S}_{t^*} \parallel (r_{t^*} \odot \mathrm{h}_{t^*-1}] + b_q), \\ \mathrm{h}_{t^*} = c_{t^*} \odot \mathrm{h}_{t^*-1} + (1 - c_{t^*}) \odot q_{t^*} \end{cases} \tag{7.8}$$

The recurrent process integrates both spatial and temporal dependencies through $\mathscr{G}$-based convolution operations at each time step t^*, enabling the model to adaptively capture evolving relationships between stream nodes. The reset gate r_{t^*} and candidate gate c_{t^*} dynamically control the flow of information within the graph structure. Mean Absolute Error (MAE) is used as the loss function in our framework, with the average loss for the multi-step prediction task calculated as follows:

$$\mathscr{L} = \mathscr{L}(\hat{\mathscr{Y}}, \mathscr{Y}) = \frac{1}{\eta I J} \sum_{t=1}^{\eta} \sum_{i=1}^{I} \sum_{j=1}^{J} | \hat{\mathscr{Y}}_{ijt} - \mathscr{Y}_{ijt} |, \tag{7.9}$$

where $\mathscr{Y} \in \mathbb{R}^{\eta \times I \times J}$ represents the prediction values, $\hat{\mathscr{Y}} \in \mathbb{R}^{\eta \times I \times J}$ represents the ground truth values. I denotes the number of streams and J denotes the output dimension.

We use rich historical multi-stream data to train, generate the multi-scale adaptive correlation graph and initialize a basic prediction model. When online testing starts, we apply online adaptation learning on this initialized model $\{\mathscr{F}_0, \mathscr{G}_0\}$ to learn new arriving samples and adapt to unknown data distribution caused by concept drift.

7.3.2 Online Adaptation Learning for Concept Drift

While existing methods offer valuable insights, they have notable limitations. During initialization, they struggle to effectively capture complex spatio-temporal dependencies between streams. During testing, they focus solely on adapting to new sample features, neglecting the challenges of historical forgetting and the reuse of historically similar drift patterns.

Our method is designed to overcome these limitations and challenges. During training, we build a high-performing base prediction model with a generalizable multi-scale adaptive correlation graph structure: $\mathscr{F}_0, \mathscr{G}_0$. In the online testing phase, we continuously monitor the model's performance and apply tailored adaptation mechanisms for drift and non-

drift scenarios. The adaptation learning process is illustrated on the right side of Fig. 7.2, with detailed calculation steps provided below.

At the start of testing, we initialize a pool and two sliding windows: a pool B for storing dynamic real-time loss values; A sliding window $\mathcal{W}^{\lambda}$ with size λ, initialized using historical features from offline training, continuously slides forward and stores new features over time; a sliding window $\mathcal{V}^{\tau}$ with size τ to store and update new multi-stream data. To detect concept drift, we set the initial loss $\mathscr{L}_0$ from the base model $\mathscr{F}0, \mathscr{G}0$ on the offline validation set as the starting value of B. As predictions proceed, new loss values at each time step $\mathscr{L}_t$ are added to B. If at any time step t, the loss $\mathscr{L}_t$ is less than the average loss in B: $\mathscr{L}_t < \overline{B}$, we assume no drift is occurring. In this case, the new data stored in $\mathcal{V}^{\tau} : \left[\mathcal{S}^{\mathcal{V}}_{t^*-\Delta+1:t^*}; \mathcal{S}^{\mathcal{V}}_{t^*+1:t^*+\eta}\right]$, along with the adaptive matrix module in $\mathscr{G}$, are used to adjust the connection weights between stream nodes in sub-graphs at a low update rate: $\{\mathscr{F}_{t-1}, \mathscr{G}_{t-1}\} \xrightarrow{\mathcal{S}^{\mathcal{V}}; e^1_{ol}} \{\mathscr{F}_t, \mathscr{G}_t\}$. The updated model then performs the prediction task for the next time step.

When $\mathscr{L}_t > \overline{B}$, concept drift is assumed to occur, and a more complex adaptation learning strategy is applied. First, we use the same multi-scale adaptive convolution used during training to the sliding window $\mathcal{W}^{\lambda}$, which contains features from both historical and new data, to generate a new graph structure G^{tem}_t. Next, to capture evolving patterns that reflect the new data distribution while retaining important historical information for global graph stability, we apply the learnable parameter σ for Exponentially Weighted Moving Average (EWMA)-style updates on sub-graphs:

$$\mathscr{G}_t = G^{tem}_t \cdot \sigma + \mathscr{G}_{t-1} \cdot (1 - \sigma) \tag{7.10}$$

This approach enables smoother adaptation of the graph structure $\mathscr{G}$ to dynamic changes in the sub-graphs.

To enhance drift adaptation, we introduce a proactive drift prediction module. This

module is motivated by the observation that, in real-world applications, drift rarely occurs in isolation—certain drift patterns may recur. While such patterns may have been encountered during training, the fast update rate of data streams and the ongoing prediction task can lead to catastrophic forgetting. To address this, when drift is detected, we match all acquired data patterns similar to the current drift, enabling drift prediction based on historical similarities.

Specifically, for each detected drift time step t, the matching process is initiated by selecting the ground truth data segment $\left[\mathscr{S}^{\mathcal{V}}_{t^*+1:t^*+\eta}\right]$ from $\mathcal{V}^{\tau}$ as the matching target. This target segment is then compared to all currently available data: $\left[\widehat{\mathscr{S}}_{t^*-\Delta+1:t^*}; \widehat{\mathscr{S}}_{t^*+1:t^*+\eta}\right]$, where the historical segments $\left[\widehat{\mathscr{S}}_{t^*-\Delta+1:t^*}\right]$ are used as the matching source. The similarity matching is performed as follows:

$$\text{sim}(t_i) = \frac{\mathscr{S}^{\mathcal{V}}_{t^*+1:t^*+\eta} \cdot \widehat{\mathscr{S}}_{t_i^*-\Delta+1:t_i^*}}{\|\mathscr{S}^{\mathcal{V}}_{t^*+1:t^*+\eta}\| \cdot \|\widehat{\mathscr{S}}_{t_i^*-\Delta+1:t_i^*}\|} \tag{7.11}$$

where t_i^* denotes each multi-step window corresponding to each time step in the acquired historical data $\widehat{\mathscr{S}}$. The top b segments with the highest similarity scores are then selected:

$$\text{top-}b \text{ segments} = \text{top-}b\left(\{\text{sim}(t_i)\}_{i=1}^{N}, b\right) \tag{7.12}$$

These selected b segments, containing potential subsequent drift patterns, are used along with data from $\mathcal{V}^{\tau}$, a new graph structure G_t^{tem} (Eq.(7.10)) and another update rate e_{ol}^2 to enhance the model's adaptation learning process:

$$\begin{cases} \mathscr{S}^{\mathcal{V}^*} = (\mathscr{S}^{\mathcal{V}}; \{\widehat{\mathscr{S}}_{i=1}^{b}\}), \\ \{\mathscr{F}_{t-1}, \mathscr{G}_{t-1}\} \xrightarrow{\mathscr{S}^{\mathcal{V}^*}; G_t^{tem}; e_{ol}^2} \{\mathscr{F}_t, \mathscr{G}_t\} \end{cases} \tag{7.13}$$

When similar drift patterns reoccur, this drift pattern matching can be seen as an active prediction for current concept drift. It also helps mitigate the issue of insufficient data within the sliding window when repetitive drift patterns emerge.

However, this process carries certain risks, such as the emergence of new drift patterns or the instability of the matched historical data. Therefore, we have also designed a dedicated dynamic validation module to ensure that the model's adaptation learning is heading in the right direction. To achieve this, we create a shadow model $\{\mathscr{F}^*_{t-1}, \mathscr{G}^*_{t-1}\}$. When drift is detected at time step t, we replicate the current model first and then carry out a synchronized adaptation process. The key difference is that the data used for the shadow model's adaptation learning is limited to the data in $\mathscr{V}^\tau$, without performing drift pattern matching:

$$\{\mathscr{F}^*_{t-1}, \mathscr{G}^*_{t-1}\} \xrightarrow{\mathscr{S}^{\mathscr{V}}; G^{tem}_t; e^2_{ol}} \{\mathscr{F}^*_t, \mathscr{G}^*_t\} \tag{7.14}$$

In this case, the shadow model will focus on the adaptation learning to new drifts. After the synchronized adaptation learning is completed, we randomly sample a subset of the validation set $\mathscr{S}^{val}$ to perform online dynamic validation. The model with better performance after validation will be used to perform the prediction at the next time step:

$$\begin{cases} \mathscr{S}^{val}_* \subset \mathscr{S}^{val}, \quad \text{where } |\mathscr{S}^{val}_*| = \kappa \cdot |\mathscr{S}^{val}| \\ \{\mathscr{F}_t, \mathscr{G}_t\} = \arg\min\left(\mathscr{L}(\mathscr{S}^{val}_*; \{\mathscr{F}^*_t, \mathscr{G}^*_t\}), \mathscr{L}(\mathscr{S}^{val}_*; \{\mathscr{F}_t, \mathscr{G}_t\})\right) \end{cases} \tag{7.15}$$

where $\mathscr{S}^{val}_*$ represents a subset of $\mathscr{S}^{val}$ selected randomly with a sampling ratio of κ. This random dynamic validation approach not only reduces the computational overhead of online adaptation learning but also helps improve the model's robustness, mitigating the risk of overfitting and ensuring a good balance between new and known data.

Overall, in MACG, whether the multi-stream is in a stable prediction state or experiencing drift, the complex spatio-temporal correlations of the multi-stream are dynamically

adjusted through various learnable modules, and the model can adapt to new samples and drifts using different adaptation learning strategies. We present the pseudo-code for the online adaptation learning of MACG, as outlined in Algorithm 4.

7.3.3 The Online Adaptation Learning of MACG

We present the pseudo-code for the online adaptation learning of MACG, as outlined in Algorithm 4.

In Steps 1 and 2, the initialization and parameter settings for the online testing are carried out. The online prediction tasks and adaptation learning processes begin in Step 3, using the base prediction model $\{\mathscr{F}_0, \mathscr{G}_0\}$. As streaming data updates and new samples arrive, the new data is stored in $\widehat{\mathscr{S}}$ (Step 4). The related windows, $\mathscr{V}^\tau$ and $\mathscr{W}^\lambda$, are updated to manage data and features, respectively (Step 5). The real-time loss of the model is calculated in Step 6, followed by updating the dynamic loss pool B in Step 7. The drift detection threshold is computed in Step 8 by averaging values stored in B.

If no drift is detected, only the data in $\mathscr{V}^\tau$ is used to drive the model and the sub-graphs updating. In this case, the adaptive matrix adjusts the correlation weights of sub-graphs at a lower update rate, e_{ol}^1 (Step 9). When drift is detected, the multi-scale adaptive convolutional layer from offline training is applied to the feature window $\mathscr{W}^\lambda$ to generate a new graph structure, G_t^{tem} (Step 11). The learnable adaptive parameter σ is then activated to update sub-graphs in Step 10. To prevent incorrect updates in subsequent steps, a shadow model $\{\mathscr{F}_{t-1}^*, \mathscr{G}_{t-1}^*\}$ is created in Step 13. Subsequently, the most similar drift patterns are searched within $\widehat{\mathscr{S}}$, and the top b historical data segments with the highest similarity scores are selected. These segments, along with the data in $\mathscr{V}^\tau$, are used for adaptation learning in Step 15. Simultaneously, adaptation learning only based on $\mathscr{V}^\tau$ for the shadow model $\{\mathscr{F}_{t-1}^*, \mathscr{G}_{t-1}^*\}$ is performed in parallel. Finally, a dynamic validation subset $\mathscr{S}_*^{val}$ is sampled to validate the performance of the two updated models.

The better-performing model is chosen (Step 16) to handle prediction tasks in the next time step.

7.4 Experiments

7.4.1 Datasets and Baselines

Datasets. We conduct comprehensive experiments on three large-scale real-world datasets to validate the effectiveness of MACG. These three datasets cover two common multi-stream scenarios: traffic and weather. A statistical summary of these datasets is provided in Table 7.2.

Given the varying complexities of the datasets, for the two traffic datasets, we use 70% of the data for initial training, 10% for cross-validation, and the remaining 20% for testing. The weather dataset is relatively simpler, so we use 50% of the data for initial training, 10% for cross-validation, and the remaining 40% for testing.

Table 7.2: Datasets Statistics Summary

DataSet	Stream Nodes	Samples	Updating Rate
WEATHER	10	29639	60min
METR-LA	207	34272	5min
PEMS-BAY	325	521160	5min

- **WEATHER** is a weather dataset collected in Beijing, China [125]. It consists of hourly weather observations recorded from January 1, 2015, to May 31, 2018. For our research, we selected three specific meteorological features: ***psur*** (surface air pressure), ***rh2m*** (relative humidity at two meters above the ground) and ***q2m*** (specific humidity at a height of two meters above the ground in grams per kilogram).

Algorithm 4 MACG Online Adaptation Learning

1: Input Multi-stream multi-step data $\mathscr{S}$ with an input length Δ and an output length η. The superscripts of $\mathscr{S}$ represent the data set sources, with $\mathrm{tr}, \mathrm{val}, \mathrm{te}, \mathrm{pre}$ denoting the training, validation, test, and prediction sets. $\widehat{\mathscr{S}}$ represents all available data for training, combining historical and new samples. To simulate streaming environments, The batch size is set to 1, with samples arriving sequentially. The learnable parameter σ in $\mathscr{G}$, sub-graphs updating rates e^1_{ol} and e^2_{ol} for non-drift and drift scenarios, and epoch counts are Q^1_m and Q^2_m. The loss function $\mathscr{L}$ is optimized using Adam [122].

2: Initial The initialized prediction model is $\{\mathscr{F}_0, \mathscr{G}_0, \Theta_0\}$. The dynamic loss pool B starts with $\mathscr{L}_0$ (the loss of base model on $\mathscr{S}^{val}$). Two sliding windows: $\mathscr{V}^\tau$ for storing new data (size τ) and $\mathscr{W}^\lambda$ for new data features (size λ), initialized with historical features from offline training. The sampling ratio κ for the dynamic validation, and the drift pattern matching considers the top b segments.

for *New samples arrive at each time step,* $t = 1, 2, 3, \ldots$ **do**

3: Multi-step Prediction:

$\mathscr{F}_t\left(\mathscr{S}^{te}_{t-\Delta+1:t}; \mathscr{G}_{t-1}, \Theta_{t-1}\right) = S^{pre}_{t+1:t+\eta}$;

if $t > 1$ **then**

4: Obtain Ground Truth: $S^{true}_{t:t+\eta-1}$**;** $\widehat{\mathscr{S}} \leftarrow \mathscr{S}^{new}$;

5: Update Sliding Windows New arriving samples and features are stored in $\mathscr{V}^\tau$ and $\mathscr{W}^\lambda$, while the old data and features will slide out of the windows;

6: Calculate Real-time Loss:

$\mathscr{L}_t = loss(F_{t-1}\left(S^{te}_{t-\Delta:t-1}; \mathscr{G}_{t-1}, \Theta_{t-1}\right), S^{true}_{t:t+\eta-1})$;

7: Update Dynamic Loss Pool:

$B = [\mathscr{L}_0, \ldots] \leftarrow \mathscr{L}_t$;

8: Concept Drift Detection: $\overline{B} = \frac{1}{t+1}\sum_{i=0}^{t} \mathscr{L}_i$

if $\mathscr{L}_t < \overline{B}$ **then**

9: Adaption Learning under Non-Drift

for $Q = 1, 2, \ldots Q^1_m$ **do**

$\mathscr{G}_{t-1}, \Theta_{t-1} = \mathrm{Adam}(\mathscr{S}^{\mathscr{V}}, \mathscr{L}_t(\mathrm{Eq.}(7.9)); e^1_{ol}), \mathscr{F}_t$;

$\mathscr{G}_t = \mathscr{G}_{t-1}$, $\mathscr{F}_t = \mathscr{F}_{t-1}$;

end

else

10: Adaption Learning under Concept Drift

for $Q = 1, 2, \ldots Q^2_m$ **do**

11: Generate A New adaptive Graph based on $\mathscr{W}^\lambda$**:** $G^{tem}_t(\mathrm{Eq.}(7.7))$**;**

12: Fuse Adaptive Graphs:

$\mathscr{G}_t = G^{tem}_t \cdot \sigma + \mathscr{G}_{t-1} \cdot (1-\sigma), (\mathrm{Eq.}(7.10)))$;

13: Generate A Shadow Model:

$\{\mathscr{F}^*_{t-1}, \mathscr{G}^*_{t-1}\} = \{\mathscr{F}_{t-1}, \mathscr{G}_{t-1}\}$;

14: Drift Patterns Matching on $\widehat{\mathscr{S}}$**:**

$\mathrm{sim}(t_i)(\mathrm{Eq.}(7.11))$; top-$b$ segments, $\mathscr{S}^{\mathscr{V}^*}(\mathrm{Eq.}(7.12))$;

15: Update the Model and Shadow Model

$\mathscr{G}_t = \mathscr{G}_{t-1}$, $\mathscr{F}_t = \mathscr{F}_{t-1}(\mathrm{Eq.}(7.13))$;

$\mathscr{G}^*_t = \mathscr{G}^*_{t-1}$, $\mathscr{F}^*_t = \mathscr{F}^*_{t-1}(\mathrm{Eq.}(7.14))$ **16: Dynamic Validation and Model Selection**

$\mathscr{S}^{val}_*; \mathscr{G}_t, \mathscr{F}_t \mathrm{Eq.}(7.15)$

end

end

end

end

- **METR-LA** is a traffic dataset that records the velocity of vehicles across the road network in LA County. The data is collected from loop detectors located at 207 sites. Spanning four months from March to June in 2012 [86].

- **PEMS-BAY** is a traffic dataset collected by the California Transportation Agencies' (CalTrans) Performance Measurement System (PeMS) [134]. It contains data from 325 sensors in the Bay Area and spans six months, from January 1 to May 31, 2017 [86].

Baselines. We select a diverse set of baseline methods, including traditional regression methods, traditional GNN-based methods, and multi-stream adaptation methods.
Traditional Regression Methods:

- **HA**: Historical Average method;

- **VAR**: Vector Auto-Regression model;

- **FNN**: Feedforward Neural Networks;

- **FC-LSTM**: Fully Connected LSTM.

Popular GNN-based Methods:

- **DCRNN** [86]: Diffusion Convolutional Recurrent Neural Network replaces the fully connected layer in GRU [133] with a diffusion convolutional layer to model spatiotemporal dependencies effectively;

- **Graph-WaveNet** [88]: Leverages node embeddings to learn an adaptable dependency matrix in the graph convolutional layer, enabling the effective capture of both temporal and spatial dependencies;

- **AGCRN** [95]: Introduces two adaptive blocks,Äînode adaptive learning and data-adaptive graph,Äîinto Adaptive Graph Convolutional Recurrent Network to model dependencies between streams;

- **STFGNN** [89]: Spatial-Temporal Fusion Graph Neural Networks use a data-driven approach to generate temporal maps during training, addressing deep correlations that spatial maps may overlook;

- **GTS** [96]: Graph for Time Series employs Gumbel sampling to construct discrete graph structures for modeling relationships among multiple data streams;

- **ASTGCN** [90]: Attention-based Spatial-Temporal Graph Convolution Networks integrate a novel attention mechanism with graph convolution to enhance performance;

- **S^{2}TAT** [135]: Synchronous Spatiotemporal Graph Transformer integrates attention mechanisms and graph convolutions to achieve simultaneous modeling of spatial and temporal data.

- **DGCRN** [91]: Dynamic Graph Convolutional Recurrent Network incorporates dynamic graphs, generated via dynamic filters and pre-defined graphs, within a dynamic convolutional recurrent framework;

Multi-stream Adaptation Methods:

- **MuNet** [111]: This method incorporates a periodically re-trained predictor and Bayesian connectors to learn and transfer new distribution information caused by concept drift. Due to its scalability constraints, we test this method on 10 streams extracted from the traffic datasets;

- **SAGN** [132]: A GNN-based framework for concept drift self-adaptation, where the adaptation process is facilitated through sub-graph online learning and periodic validation and reselection of stored models.

- **MSGR** [140]: A multi-stream adaptation method based on graph structure constraints, using the nearest neighbours calculated by the sliding window as a regularization auxiliary for the self-adaptation process.

Table 7.3: Benchmark Comparison of MACG and Baseline Methods on Real-world Datasets

	Step	Metric	HA	VAR	SVR	FC-LSTM	Graph WaveNet	DGCRN	ASTGCN	AGCRN	DCRNN	GTS	STFGNN	S2TAT	MuNet	SAGN	MSGR	***MACG***
METR-LA	Step 3	MAE	4.16	4.42	3.99	3.44	2.69	2.62	4.86	2.87	2.77	2.64	3.23	2.78	10.02	2.46	2.42	**2.40**
		RMSE	7.80	7.89	8.45	6.30	5.15	5.01	9.27	5.58	5.38	4.95	7.43	5.43	10.02	4.41	4.34	**4.31**
		MAPE	13.00%	10.20%	9.30%	9.60%	6.90%	6.63%	9.21%	7.70%	7.30%	6.80%	7.82%	7.38%	15.73%	6.41%	6.26%	**6.15%**
	Step 6	MAE	4.16	5.41	5.05	3.77	3.07	2.99	5.43	3.23	3.15	3.01	4.02	3.10	10.06	2.79	2.72	**2.70**
		RMSE	7.80	9.13	10.87	7.23	6.22	6.05	10.61	6.58	6.45	5.85	9.49	6.39	10.06	5.15	5.04	**5.02**
		MAPE	13.00%	12.7	12.10%	10.90%	8.37%	8.02%	10.13%	9.00%	8.80%	8.20%	10.00%	8.70%	15.79%	7.64%	7.41%	**7.27%**
	Step 12	MAE	4.16	6.52	6.72	4.49	3.53	3.44	6.51	3.62	3.60	3.41	5.05	3.43	10.13	3.16	3.07	**3.04**
		RMSE	7.80	10.11	13.76	8.69	7.37	7.19	12.52	7.51	7.59	6.74	11.67	7.32	10.13	5.91	5.77	**5.73**
		MAPE	13.00%	15.80%	16.70%	13.20%	10.10%	9.73%	11.64%	10.38%	10.50%	9.90%	12.97%	10.02%	15.97%	9.10%	8.79%	**8.67%**
PEMS-BAY	Step 3	MAE	2.88	1.74	1.85	2.05	1.30	1.28	1.52	1.37	1.38	1.32	1.39	1.33	5.26	1.30	1.29	**1.28**
		RMSE	5.59	3.16	3.59	4.19	2.74	2.69	3.13	2.87	2.95	2.62	2.90	2.89	5.26	2.43	2.43	**2.40**
		MAPE	6.80%	3.60%	3.80%	4.80%	2.73%	2.66%	3.22%	2.94%	2.90%	2.80%	2.96%	2.83%	18.36%	2.75%	2.71%	**2.67%**
	Step 6	MAE	2.88	2.32	2.48	2.20	1.63	1.59	2.01	1.69	1.74	1.64	1.77	1.62	5.31	1.60	1.58	**1.56**
		RMSE	5.59	4.25	5.18	4.55	3.70	3.63	4.27	3.85	3.97	3.41	3.97	3.73	5.31	3.11	3.09	**3.05**
		MAPE	6.80%	5.00%	5.50%	5.20%	3.67%	3.55%	4.48%	3.87%	3.90%	3.60%	4.02%	3.67%	18.46%	3.61%	3.54%	**3.47%**
	Step 12	MAE	2.88	2.93	3.28	2.37	1.95	1.89	2.61	1.96	2.07	1.91	2.15	1.85	5.46	1.85	1.81	**1.80**
		RMSE	5.59	5.44	7.08	4.96	4.52	4.42	5.42	4.54	4.74	3.97	4.92	4.30	5.46	3.61	3.55	**3.55**
		MAPE	6.80%	6.50%	8.00%	5.70%	4.63%	4.43%	6.00%	4.64%	4.90%	4.40%	5.17%	4.31%	18.71%	4.37%	4.26%	**4.22%**
WEATHER (psur)	Step 3	MAE	28.52	1.56	1.62	2.06	1.00	0.94	1.52	1.03	1.06	1.04	1.44	1.26	3.42	0.96	0.93	**0.91**
		RMSE	39.96	4.60	5.21	5.00	4.80	4.33	4.50	4.77	4.81	2.86	5.00	5.00	3.42	1.29	1.25	**1.23**
		MAPE	3.00%	0.16%	0.16%	0.21%	0.10%	0.10%	0.14%	0.11%	0.11%	0.11%	0.15%	0.13%	0.35%	0.10%	0.09%	**0.09%**
	Step 6	MAE	28.52	2.30	2.57	2.86	1.44	1.56	2.15	1.48	1.51	1.55	2.03	1.83	3.45	1.43	1.38	**1.36**
		RMSE	39.96	5.36	6.29	5.80	5.13	5.09	5.32	5.11	5.23	3.39	5.33	4.74	3.45	1.79	1.73	**1.72**
		MAPE	3.00%	0.23%	0.26%	0.29%	0.15%	0.16%	0.22%	0.15%	0.16%	0.16%	0.21%	0.19%	0.35%	0.15%	0.14%	**0.14%**
	Step 12	MAE	28.52	3.09	3.58	3.71	2.25	2.71	2.90	2.31	2.37	2.49	2.79	2.59	3.58	2.29	2.21	**2.19**
		RMSE	39.96	6.02	7.21	6.54	5.57	6.11	5.94	5.60	5.24	4.30	5.85	5.66	3.58	2.67	2.56	**2.56**
		MAPE	3.00%	0.31%	0.36%	0.38%	0.23%	0.28%	0.30%	0.24%	0.24%	0.25%	0.28%	0.26%	0.36%	0.23%	0.23%	**0.22%**
WEATHER (q2m)	Step 3	MAE	5.15	0.61	0.25	0.30	0.56	0.56	0.61	0.59	0.59	0.60	0.64	0.63	1.33	0.56	0.55	**0.54**
		RMSE	6.29	1.04	0.58	0.58	1.00	1.01	1.02	1.03	1.03	0.88	1.06	1.12	1.33	0.69	0.68	**0.67**
		MAPE	191.12%	12.64%	13.32%	19.28%	10.38%	10.89%	13.39%	11.20%	11.20%	11.26%	12.22%	12.35%	42.04%	10.72%	10.59%	**10.45%**
	Step 6	MAE	5.15	0.86	0.38	0.43	0.79	0.81	0.86	0.82	0.82	0.84	0.89	0.84	1.35	0.79	0.77	**0.76**
		RMSE	6.29	1.34	0.74	0.73	1.30	1.34	1.34	1.33	1.34	1.17	1.38	1.34	1.35	0.93	0.91	**0.90**
		MAPE	191.12%	18.84	21.32%	29.07%	15.30%	16.61%	17.63%	16.13%	16.45%	16.47%	16.98%	16.10%	42.55%	16.02%	15.60%	**15.49%**
	Step 12	MAE	5.15	1.17	0.53	0.59	1.10	1.17	1.16	1.16	1.16	1.16	1.16	1.15	1.55	1.10	1.06	**1.05**
		RMSE	6.29	1.72	0.89	0.88	1.70	1.79	1.72	1.74	1.77	1.53	1.73	1.74	1.55	1.25	1.20	**1.19**
		MAPE	191.12%	27.73%	32.70%	42.29%	23.33%	25.83%	25.11%	25.00%	24.98%	24.43%	24.46%	24.88%	46.65%	24.24%	23.20%	**23.05%**
WEATHER (rh2m)	Step 3	MAE	20.41	7.43	7.79	7.82	5.78	7.15	6.75	6.44	5.78	5.56	6.70	6.89	13.20	5.66	5.35	**5.29**
		RMSE	23.65	10.24	11.51	11.13	8.67	10.42	9.41	9.47	8.62	8.10	9.81	9.75	13.20	6.97	6.66	**6.62**
		MAPE	56.37%	19.22%	23.97%	24.26%	13.76%	15.23%	16.31%	15.40%	13.73%	13.17%	16.09%	16.30%	36.89%	13.09%	12.98%	**12.61%**
	Step 6	MAE	20.41	12.09	12.85	12.69	8.21	13.51	10.03	9.57	8.27	8.03	9.80	9.78	13.26	8.46	7.63	**7.56**
		RMSE	23.65	15.30	16.87	16.37	11.98	18.30	13.23	13.52	11.98	11.08	13.65	13.46	13.26	9.97	9.11	**9.07**
		MAPE	56.37%	33.41%	42.44%	42.60%	19.92%	22.05%	25.55%	24.11%	20.06%	19.56%	24.78%	24.04%	37.09%	19.69%	19.17%	**18.53%**
	Step 12	MAE	20.41	16.32	13.75	14.21	10.90	24.93	11.99	11.38	11.18	10.92	11.63	11.66	13.48	11.71	10.28	**10.11**
		RMSE	23.65	19.76	18.26	18.01	15.30	30.86	15.70	15.79	15.75	14.48	16.03	16.05	13.54	13.35	11.83	**11.68**
		MAPE	56.37%	47.83%	42.23%	46.98%	27.56%	27.35%	31.21%	29.34%	28.08%	27.93%	30.65%	30.14%	37.67%	28.36%	27.15%	**26.12%**

7.4.2 Experimental Setup

Platform. The model implementation is carried out using PyTorch 1.7.1. All experiments are conducted on a server equipped with two NVIDIA A100 Tensor Core GPUs (80GB memory each) and an Intel(R) Xeon(R) Gold 6342 CPU running at 2.80GHz.

Parameter settings. In the model training phase, the Gumbel sampling parameter δ is set to 0.9. The initial learning rate is 0.005 with 200 epochs for the traffic datasets and 0.001 with 300 epochs for the weather dataset. The diffusion step number is set to 3 for METR-LA, WEATHER (***psur***), and WEATHER (***rh2m***), and 2 for PEMS-BAY and WEATHER (***q2m***).

For online adaptation, the sub-graph learning rate under non-drift conditions is 0.000005 for METR-LA, PEMS-BAY, WEATHER (***psur***), and WEATHER (***q2m***), and 0.0000005 for WEATHER (***rh2m***). During drift, the learning rate is 0.00001 for METR-LA and PEMS-BAY, 0.000005 for WEATHER (***psur***) and WEATHER (***rh2m***), and 0.0000001 for WEATHER (***q2m***). The drift pattern match size is 2 for METR-LA and WEATHER (***rh2m***), and 127 for the remaining datasets. The validation ratio is set to 0.8.

Evaluation Metrics. The performance of MACG and all comparison methods is assessed using mean absolute error (MAE), mean absolute percentage error (MAPE), and root mean square error (RMSE). The predicted values and ground truth values are first converted back to their original scales before calculating the evaluation metrics.

7.4.3 Experimental Results

We evaluate MACG using three real-world datasets of different scales, each with varying numbers of stream nodes. The two traffic datasets, METR-LA and PEMS-BAY, have a larger number of stream nodes, while the weather dataset has fewer. This highlights the adaptability of MACG to data streams of different scales.

Table 7.3 presents the benchmark comparison results. Our method, MACG, outper-

forms baseline models in multi-stream, multi-step prediction tasks. The METR-LA and WEATHER (***rh2m***) datasets experience more frequent drifts, while other datasets have more stable data distributions. Especially for WEATHER (***q2m***), the changes in this dataset are very regular, so some traditional machine learning models also perform well in the tests. Across the four multi-stream scenarios and three datasets, MACG improves prediction performance over the state-of-the-art traditional GNN-based model by 9.52%, 2.44%, 15.19%, 4.09%, and 6.36%, respectively. It also surpasses the state-of-the-art GNN-based multi-stream adaptation model by 0.25%, 1.28%, 2.13%, 1.26%, and 1.29%, respectively. Thanks to the model's ability to capture complex spatio-temporal correlations across multiple time scales and the active drift pattern matching, we achieve significant improvements over traditional GNN models, and further improve adaptation performance over existing multi-stream adaptation methods. Meanwhile, MACG can maintain strong performance even in cases of less severe drift fluctuations.

7.4.4 Ablation Study

We conduct ablation experiments on the METR-LA dataset to verify the effectiveness of each component in our framework.

w/o ol refers to the complete removal of the online adaptation module. In this case, the model performs the same prediction task as traditional models, performing static prediction under offline mode without considering concept drift in the test set. To evaluate the high generalizability of our initialized model, including the multi-stream correlation graph structure, we compare its performance with the initialized models of state-of-the-art multi-stream adaptation frameworks "SAGN ol" and "MSGR ol". From Table 7.4, it can be observed that our initialized model is still better equipped to adapt to unknown concept drift in the test set compared to existing models.

w/o adp removes the concept drift detection and adaptation modules. The model relies

Table 7.4: Ablation Study

	Step	Metric	SAGN o/l	MSGR o/l	w/o ol	w/o adp	***MACG***
METR-LA	Step 3	MAE	2.64	2.60	2.59	2.41	**2.40**
		RMSE	4.95	4.42	4.86	4.34	**4.31**
		MAPE	6.80%	6.86%	6.54%	6.25%	**6.15%**
	Step 6	MAE	3.01	2.97	2.94	2.73	**2.70**
		RMSE	5.85	5.78	5.75	5.07	**5.02**
		MAPE	8.20%	8.36%	7.81%	7.45%	**7.27%**
	Step 12	MAE	3.41	3.35	3.31	3.10	**3.04**
		RMSE	6.74	6.65	6.59	5.84	**5.73**
		MAPE	9.90%	10.07%	9.33%	8.83%	**8.67%**

only on sub-graph updates driven by new samples to adapt to drift. From Table 7.4, the results show that the model still performs better than the static model, proving the need for a multi-stream concept drift adaptation framework. However, without targeted adaptation strategies, its performance still has room for improvement.

7.4.5 Parameters Sensitivity

We construct parameter sensitivity experiments on four parameters in online adaptation learning: the sub-graph learning rate e_{ol}^1, e_{ol}^2 under non-drift and drift cases, the drift pattern match size b and the dynamic validation set sampling ratio κ. We use MAE as the metric in these tests, and the experimental results are shown in Fig. 7.3.

Since we have constructed a highly generalizable model and correlation graph structure during the initialization phase, which already has some ability to adapt to concept drift, the update rates used for sub-graph updates during online adaptation learning should not be set too high to avoid overfitting to new samples or matched drift patterns. We use two different update rates, e_{ol}^1 and e_{ol}^2 (Fig. 7.3 (a), (b)), to maintain model performance

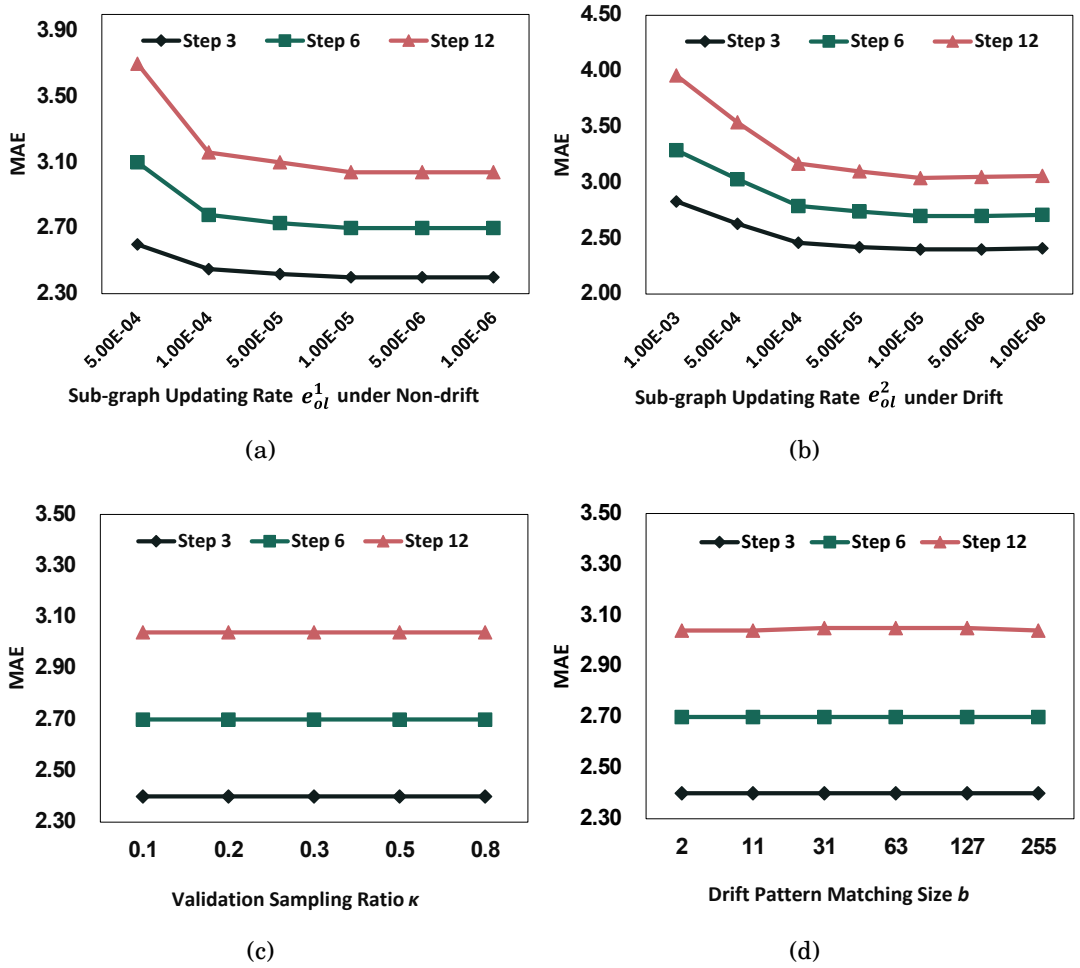


Figure 7.3: The effect of different parameters in online adaptation learning.

stability during non-drift periods while allowing the model to adapt smoothly to new sample distributions during the drift.

The parameter κ represents the proportion of randomly selected samples from the validation set during dynamic validation. The model is not sensitive to this parameter (Fig. 7.3 (c)) because the drift fluctuation is not large, and the large sample size provides enough data for validation. We recommend increasing this proportion when data fluctuations are large, or the validation set is small to ensure robust validation.

The matching parameter b for the drift pattern represents the amount of data extracted

from historical data that is highly similar to the current drift pattern. Here, we observe that the model is not very sensitive to this parameter (Fig. 7.3 (d)). As the data stream updates over time, the drift pattern matching is actually based on local drift features, rather than requiring large amounts of data to drive model updates. Furthermore, if the distribution in the test set differs significantly from that in the training set, our dynamic validation will prevent such matching.

7.5 Summary

This chapter introduces a multi-scale adaptive convolutional graph framework, MACG, designed for multi-stream concept drift adaptation. The framework is developed around two key aspects to enhance the model's ability to adapt to concept drift: the generalization performance of the initialized model and the ability to proactively predict drift during online testing.

For the former, we propose a novel graph neural network architecture based on a multi-stream, multi-scale adaptive correlation graph, replacing the traditional reliance on pre-defined graphs. By performing multi-scale convolutions and multi-sampling on historical data features, the framework initializes a highly generalizable multi-stream correlation graph while capturing complex spatio-temporal correlations and dynamic patterns between streams.

For the latter, the model's real-time performance serves as an indicator for drift detection, triggering different adaptive learning mechanisms for online adaptation. When no drift occurs, new samples drive sub-graph updates. When drift occurs, a more sophisticated adaptive learning mechanism is employed. Multi-scale adaptive convolution layers are reused on the feature sliding window to generate a new graph, which is adaptively fused with the previous graph. Simultaneously, potential drift patterns are matched from historical data. Finally, the fused graph, matched drift patterns, and new samples

jointly drive sub-graph updates, enabling the model to adapt effectively. This mechanism predicts drift trends to a certain extent while mitigating catastrophic forgetting. Extensive experiments demonstrate the effectiveness and outstanding performance of MACG.

CHAPTER 8

CONCLUSIONS AND FUTURE STUDY

This chapter concludes the thesis and suggests directions for future research.

8.1 Conclusions

Recent research on concept drift has achieved success in many fields and has demonstrated significant advantages in various practical applications. However, these existing works also have certain limitations. Many frameworks and methods focus on single-stream scenarios, often neglecting multi-stream contexts. In fact, multi-streams not only have broader application potential but also present more challenging obstacles in developing corresponding frameworks. To fill this gap, this thesis focuses on autonomous Learning for Multiple Data Streams under Concept Drift. There are five main challenges addressed in this work: i) identifying the concept drift problem on multiple data streams; ii) recognizing the drift relationship between multiple data streams according to the concept drift in each data stream; iii) solving concurrent drift in multiple data streams with considering the drift relationship between streams; iV) solving delayed drift in mul-

tiple data streams with considering the relationship between streams and v) designing autonomous learning algorithms for multiple data streams under concurrent/delayed drift.

To address the challenges outlined above, this thesis presents five specific research questions along with their corresponding objectives. The key findings of this study are summarized as follows:

1. The theoretical foundation is built for identifying the concept drift problem and recognizing the drift relationship between multiple data streams according to the concept drift in each data stream. (to achieve RO 1)

2. We propose a multi-stream adaptive framework targeted to the concurrent drift problem based on the correlation between streams. A novel Bayesian-based connector is designed to learn the drift information from the base stream without a separate adaptive method required. (to achieve RO 2)

3. We propose a self-adaptation framework based on graph neural networks. The correlation in multi-stream is presented by a graph structure. Any type of drift can be adapted during the online learning phase and multiple GNNs perform prediction tasks by rolling strategy. (to achieve RO 3)

4. We design a concept drift self-adaptation framework for multi-stream based on dynamic graph regularization. It takes full advantage of the original graph structure, which achieves self-adaptation by error-based detection and applies corresponding dynamic graph regularization weight based on the drift level. (to achieve RO 4)

5. We consider stricter multi-stream model initialization conditions and extend the framework's adaptability to concept drift from a lifelong learning perspective. The model can be initialized by just a small amount of historical data and performs stable prediction during long-term autonomous learning. (to achieve RO 4)

6. We approach multi-step prediction by utilizing multi-scale convolutional time windows to capture the dynamic correlation and data feature of multi-stream. The initialized model with high generalizability balances the prediction accuracy for both short-term and long-term horizons. Also, the proactive drift matching module is employed to achieve more accurate adaptation performance and avoid the risk of catastrophic forgetting. (to achieve RO 4)

Furthermore, based on the outcomes of this thesis, several valuable insights have been gained regarding the strengths and limitations of leveraging inter-stream correlations to address the concept drift problem in multi-stream scenarios. A summary is provided below:

Strengths:

1. **Identification and Definition of Concept Drift in Multi-Stream:** The foundation for addressing concept drift in multi-stream settings lies in effectively identifying and defining such drifts. A fundamental challenge in multi-stream concept drift is the inherent difference between handling drifts across multiple streams and focusing solely on a single data stream. Multi-stream concept drift exhibits greater complexity, often accompanied by dynamic changes in inter-stream correlations triggered by the drift.

2. **Recognition and Representation of Multi-Stream Correlations:** One of the significant challenges in multi-stream concept drift is identifying and representing inter-stream correlations, which are essential for constructing data- and correlation-driven drift learning models. Graph structures, a traditional tool in machine learning, can model deep spatiotemporal correlations among stream objects. Integrating graph structures into a concept drift learning framework addresses the static limitations of traditional graphs, enabling the automatic adjustment

and adaptation to new multi-stream data distributions that encompass novel correlations.

3. **Enhanced Flexibility:** Building flexible graph neural network-based models removes reliance on the volume of initial training data or predefined graphs. By leveraging a learnable and adaptive framework, these models maintain the ability to capture and adapt to evolving data distributions over extended periods of online learning. This automated adaptation process enables rapid responses to drifts of varying magnitudes, benefiting multi-stream prediction tasks by ensuring sustained precision and stability.

4. **Improved Generalization:** Traditional machine learning approaches focus on constructing static models, which are less suitable for concept drift scenarios. However, the generalization capability of the initial model remains critical for adapting to unknown data distributions. Employing a highly generalized correlation-driven graph structure to power multi-stream concept drift learning models enables faster and more accurate adaptation to inconsistent data distributions during online learning. This approach also enhances the model's capacity to learn and integrate new distribution information.

Limitations:

1. **Precise Drift Detection Mechanism:** While error-rate-based drift detection is quick and effective, the complexity of multi-stream scenarios demands a more precise detection mechanism. This is particularly necessary to identify drift in specific streams or across multiple streams, requiring more sophisticated detection modules.

2. **Increased Complexity:** Concept drift learning models based on improved graph neural networks must address both offline initialization and online adaptation.

Consequently, as the scale of multi-stream data grows significantly, the computational cost of the model increases accordingly.

8.2 Future Study

Although our current work has partially alleviated the issue of multi-stream concept drift in some scenarios, it remains unexplored in more complex real-world settings. For future research, This thesis identifies further investigate this topic based on the limitations of the present work, focusing on the following aspects:

1. *Enhanced Drift Detection*

   Future research will focus on improving the drift detection module within the framework. This will enable the model to precisely pinpoint the specific locations of drift within multi-stream environments and help differentiate between drift and noise. By doing so, the model can avoid erroneous updates and reduce unnecessary computational overhead.

2. *Smarter Adaptation Strategies*

   In the current work, the overall drift level of multiple streams is linked to parameters such as the model's update rate, allowing the model to adjust its adaptation level automatically. In future work, with a more refined drift detection mechanism, the model can focus more on streams with significant drift. This targeted approach will enable the subgraph updating process in the multi-stream relational graph structure to adjust correlation weights more accurately.

3. *Framework Scalability*

   The current framework is designed for predictive tasks in multi-stream environments. However, certain dynamic multi-stream scenarios, such as when the number

of data streams changes, cannot be directly addressed. In future work, we aim to design the framework with this challenge in mind to enhance its adaptability.

4. *Increased Automation*

   While the current framework achieves automatic concept drift adaptation through a highly generalized initialization model and drift-level-linked adaptation mechanisms, many parameters still require manual tuning during training. In future work, we will aim to further reduce the reliance on manual parameter adjustments, enhancing the model's automation and self-learning capabilities.

BIBLIOGRAPHY

[1] Ahmed Oussous, Fatima-Zahra Benjelloun, Ayoub Ait Lahcen, and Samir Belfkih.
Big data technologies: A survey.
J. King Saud Univ. - Comput. Inf. Sci., 30(4):431–448, 2018.

[2] Seref Sagiroglu and Duygu Sinanc.
Big data: A review.
In *Proc. CTS*, pages 42–47, San Diego, CA, USA May 20-24, 2013. IEEE.

[3] Bartosz Krawczyk, Leandro L Minku, João Gama, Jerzy Stefanowski, and Michał Woźniak.
Ensemble learning for data stream analysis: A survey.
Inf. Fusion, 37:132–156, 2017.

[4] Jie Lu, Anjin Liu, Yiliao Song, and Guangquan Zhang.
Data-driven decision support under concept drift in streamed big data.
Complex & Intell. Syst., 6(1):157–163, 2020.

[5] Hossein Ghomeshi, Mohamed Medhat Gaber, and Yevgeniya Kovalchuk.
Eacd: evolutionary adaptation to concept drifts in data streams.
Data Min. Knowl. Discov., 33(3):663–694, 2019.

[6] Z. Yang, S. Al-Dahidi, P. Baraldi, E. Zio, and L. Montelatici.
A novel concept drift detection method for incremental learning in nonstationary environments.

IEEE Trans. Neural Netw. & Learn. Syst., 31(1):309–320, 2020.

[7] Shuo Wang, Leandro L Minku, and Xin Yao.
A systematic study of online class imbalance learning with concept drift.
IEEE Trans. Neural Netw. & Learn. Syst., 29:4802–4821, 2018.

[8] Lior Cohen, Gil Avrahami-Bakish, Mark Last, Abraham Kandel, and Oscar Kipersztok.
Real-time data mining of non-stationary data streams from sensor networks.
Inf. Fusion, 9(3):344–353, 2008.

[9] Amr Abdullatif, Francesco Masulli, and Stefano Rovetta.
Clustering of nonstationary data streams: A survey of fuzzy partitional methods.
Data Min. Knowl. Discov., 8(4):e1258, 2018.

[10] Alexey Tsymbal.
The problem of concept drift: definitions and related work.
Comput. Sci. Dept., Trinity Coll. Dublin, 106(2):58, 2004.

[11] Gregory Ditzler, Manuel Roveri, Cesare Alippi, and Robi Polikar.
Learning in nonstationary environments: A survey.
IEEE Comput. Intell. Mag., 10(4):12–25, 2015.

[12] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang.
Learning under concept drift: A review.
IEEE Trans. Knowl. Data Eng., 31(12):2346–2363, 2018.

[13] Jicheng Shan, Hang Zhang, Weike Liu, and Qingbao Liu.
Online active learning ensemble framework for drifted data streams.
IEEE Trans. Neural Netw. & Learn. Syst., 30(2):486–498, 2018.

[14] Anjin Liu, Jie Lu, and Guangquan Zhang.
Concept drift detection via equal intensity k-means space partitioning.
IEEE Trans. Cybern., 51(6):3198–3211, 2020.

[15] Roberto Souto Maior Barros and Silas Garrido T Carvalho Santos.
A large-scale comparison of concept drift detectors.
Inf. Sci., 451:348–370, 2018.

[16] Yiliao Song, Guangquan Zhang, Haiyan Lu, and Jie Lu.
A self-adaptive fuzzy network for prediction in non-stationary environments.
In *Proc. FUZZ-IEEE*, pages 1–8, Rio de Janeiro, Brazil, Jul. 8-13, 2018. IEEE.

[17] Yiliao Song, Guangquan Zhang, Haiyan Lu, and Jie Lu.
A noise-tolerant fuzzy c-means based drift adaptation method for data stream regression.
In *Proc. FUZZ-IEEE*, pages 1–6. IEEE, 2019.

[18] Oscar Koller, Cihan Camgoz, Hermann Ney, and Richard Bowden.
Weakly supervised learning with multi-stream cnn-lstm-hmms to discover sequential parallelism in sign language videos.
IEEE Trans. Pattern Anal. Mach. Intell., 2019.

[19] Michael D Jones, Holly L Peterson, Jonathan J Pierce, Nicole Herweg, Amiel Bernal, Holly Lamberta Raney, and Nikolaos Zahariadis.
A river runs through it: A multiple streams meta-review.
Policy Stud. J., 44(1):13–36, 2016.

[20] Joao Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues.
Learning with drift detection.
In *Brazilian Symp. Artif. Intell.*, pages 286–295. Springer, 2004.

[21] Manuel Baena-Garcıa, José del Campo-Ávila, Raúl Fidalgo, Albert Bifet, R Gavalda, and Rafael Morales-Bueno.
Early drift detection method.
In *Proc. KDDS*, volume 6, pages 77–86, 2006.

[22] Kyosuke Nishida and Koichiro Yamauchi.
Detecting concept drift using statistical testing.
In *Int. Conf. Discov. Sci.*, pages 264–269, Sendai, Japan, October 1-4, 2007. Springer.

[23] Joao Gama and Gladys Castillo.
Learning with local drift detection.
In *Int. Conf. Adv. Data Mining Appl.*, pages 42–55, Xi'an, China, August 14-16, 2006. Springer.

[24] Parinaz Sobhani and Hamid Beigy.
New drift detection method for data streams.
In *Int. Conf. Adapt. Intell. Syst.*, pages 88–97. Springer, 2011.

[25] Gordon J Ross, Niall M Adams, Dimitris K Tasoulis, and David J Hand.
Exponentially weighted moving average charts for detecting concept drift.
Pattern Recogn. Lett., 33(2):191–198, 2012.

[26] Roberto SM Barros, Danilo RL Cabral, Paulo M Gonçalves Jr, and Silas GTC Santos.
Rddm: Reactive drift detection method.
Expert Syst. Appl., 90:344–355, 2017.

[27] Isvani Frias-Blanco, José del Campo-Ávila, Gonzalo Ramos-Jimenez, Rafael Morales-Bueno, Agustin Ortiz-Diaz, and Yaile Caballero-Mota.

Online and non-parametric drift detection methods based on hoeffding,Äôs bounds. *IEEE Trans. Knowl. Data Eng.*, 27(3):810–823, 2014.

[28] Ali Pesaranghader and Herna L Viktor.
Fast hoeffding drift detection method for evolving data streams.
In *Joint Eur. Conf. Mach. Learn. Knowl. Discov. Databases*, pages 96–111. Springer, 2016.

[29] Anjin Liu, Guangquan Zhang, and Jie Lu.
Fuzzy time windowing for gradual concept drift adaptation.
In *Proc. FUZZ-IEEE*, pages 1–6, 2017.

[30] Shuliang Xu and Junhong Wang.
Dynamic extreme learning machine for data stream classification.
Neurocomputing, 238:433–449, 2017.

[31] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew.
Extreme learning machine: theory and applications.
Neurocomputing, 70(1-3):489–501, 2006.

[32] Ning Lu, Guangquan Zhang, and Jie Lu.
Concept drift detection via competence models.
Artif. Intell., 209:11–28, 2014.

[33] Tamraparni Dasu, Shankar Krishnan, Suresh Venkatasubramanian, and Ke Yi.
An information-theoretic approach to detecting changes in multi-dimensional data streams.
In *Proc. Symp. Interface Stat. Comput. Sci. Appl.*, Pasadena, May 24-27, 2006. Citeseer.

[34] Junming Shao, Zahra Ahmadi, and Stefan Kramer.

Prototype-based learning on concept-drifting data streams.
In *Proc. KDD*, pages 412–421, 2014.

[35] Ning Lu, Jie Lu, Guangquan Zhang, and Ramon Lopez De Mantaras.
A concept drift-tolerant case-base editing technique.
Artif. Intell., 230:108–133, 2016.

[36] Daniel Kifer, Shai Ben-David, and Johannes Gehrke.
Detecting change in data streams.
In *Proc. VLDB*, volume 4, pages 180–191. Toronto, Canada, 2004.

[37] Albert Bifet and Ricard Gavalda.
Learning from time-changing data with adaptive windowing.
pages 443–448. SIAM, 2007.

[38] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, and Ricard Gavalda.
Improving adaptive bagging methods for evolving data streams.
In *Asian Conf. Mach. Learn.*, pages 23–37, Nanjing, China, November 2-4, 2009. Springer.

[39] Albert Bifet and Ricard Gavalda.
Adaptive learning from evolving data streams.
In *Int. Conf. Discov. Sci.*, pages 249–260, Lyon, France, August 31-September 2, 2009. Springer.

[40] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Richard Kirkby, and Ricard Gavalda.
New ensemble methods for evolving data streams.
In *Proc. KDD*, pages 139–148, Paris, France, June 28-July 1, 2009.

[41] Heitor M Gomes, Albert Bifet, Jesse Read, Jean Paul Barddal, Fabrício Enembreck, Bernhard Pfharinger, Geoff Holmes, and Talel Abdessalem.
Adaptive random forests for evolving data stream classification.
Mach. Learn., 106(9):1469–1495, 2017.

[42] Xiuyao Song, Mingxi Wu, Christopher Jermaine, and Sanjay Ranka.
Statistical change detection for multi-dimensional data.
In *Proc. KDD*, pages 667–676, San Jose, California, USA, August 12-15, 2007.

[43] Li Bu, Cesare Alippi, and Dongbin Zhao.
A pdf-free change detection test based on density difference estimation.
IEEE Trans. Neural Netw. & Learn. Syst., 29(2):324–334, 2016.

[44] Li Bu, Dongbin Zhao, and Cesare Alippi.
An incremental change detection test based on density difference estimation.
IEEE Trans. Syst. Man Cybern. Syst., 47(10):2714–2726, 2017.

[45] Feng Gu, Guangquan Zhang, Jie Lu, and Chin-Teng Lin.
Concept drift detection based on equal density estimation.
In *Proc. IJCNN*, pages 24–30, Vancouver, Canada, July 24-29, 2016. IEEE.

[46] Anjin Liu, Yiliao Song, Guangquan Zhang, and Jie Lu.
Regional concept drift detection and density synchronized drift adaptation.
In *Proc. IJCAI*, 2017.

[47] Cesare Alippi, Giacomo Boracchi, and Manuel Roveri.
Just-in-time classifiers for recurrent concepts.
IEEE Trans. Neural Netw. & Learn. Syst., 24(4):620–634, 2013.

[48] Heng Wang and Zubin Abraham.
Concept drift detection for streaming data.

In *Proc. IJCNN*, pages 1–9, Killarney, Ireland, July 12-17, 2015. IEEE.

[49] Cesare Alippi, Giacomo Boracchi, and Manuel Roveri.
Hierarchical change-detection tests.
IEEE Trans. Neural Netw. & Learn. Syst., 28(2):246–258, 2016.

[50] Shujian Yu and Zubin Abraham.
Concept drift detection with hierarchical hypothesis testing.
In *Proc. SIAM Int. Conf. Data Min.*, pages 768–776, Pittsburgh, Pennsylvania, USA, July 10-14, 2017. SIAM.

[51] Yuhong Zhang, Guang Chu, Peipei Li, Xuegang Hu, and Xindong Wu.
Three-layer concept drifting detection in text data streams.
Neurocomputing, 260:393–403, 2017.

[52] Zhe Yang, Sameer Al-Dahidi, Piero Baraldi, Enrico Zio, and Lorenzo Montelatici.
A novel concept drift detection method for incremental learning in nonstationary environments.
IEEE Trans. Neural Netw. & Learn. Syst., 31(1):309–320, 2019.

[53] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia.
A survey on concept drift adaptation.
ACM Comput. Surv., 46(4):1–37, 2014.

[54] Dong Liu, YouXi Wu, and He Jiang.
Fp-elm: An online sequential learning algorithm for dealing with concept drift.
Neurocomputing, 207:322–334, 2016.

[55] Yiliao Song, Jie Lu, Anjin Liu, Haiyan Lu, and Guangquan Zhang.
A segment-based drift adaptation method for data streams.

IEEE Trans. Neural Netw. & Learn. Syst., 33(9):4876–4889, 2022.

[56] Fan Dong, Jie Lu, Yiliao Song, Feng Liu, and Guangquan Zhang.
A drift region-based data sample filtering method.
IEEE Trans. Cybern., 52(9):9377–9390, 2022.

[57] Cesare Alippi, Giacomo Boracchi, and Manuel Roveri.
A just-in-time adaptive classification system based on the intersection of confidence intervals rule.
Neural Netw., 24(8):791–800, 2011.

[58] Cesare Alippi, Giacomo Boracchi, and Manuel Roveri.
Just-in-time ensemble of classifiers.
In *Proc. IJCNN*, pages 1–8, Brisbane, QLD, Australia, June 10-15, 2012. IEEE.

[59] Ivan Koychev.
Gradual forgetting for adaptation to concept drift.
In *Proc. ECAI Workshop Curr. Issues Spatio-Temporal Reasoning*, Berlin, Germany, August 20-25, 2000.

[60] Edith Cohen and Martin Strauss.
Maintaining time-decaying stream aggregates.
In *Proc. ACM SIGMOD-SIGACT-SIGART Symp. Principles Database Syst.*, pages 223–233, San Diego, California, June 9-11, 2003.

[61] Cesare Alippi, Giacomo Boracchi, and Manuel Roveri.
Just in time classifiers: Managing the slow drift case.
In *Proc. IJCNN*, pages 114–120, Atlanta, Georgia, USA. June 14-19, 2009. IEEE.

[62] Jeffrey S Vitter.
Random sampling with a reservoir.

ACM Trans. Math. Softw., 11(1):37–57, 1985.

[63] Charu C Aggarwal.
On biased reservoir sampling in the presence of stream evolution.
In *Proc. VLDB*, pages 607–618, Seoul, Korea, September 12-15, 2006.

[64] Pedro Domingos and Geoff Hulten.
Mining high-speed data streams.
In *Proc. KDD*, pages 71–80, 2000.

[65] Geoff Hulten, Laurie Spencer, and Pedro Domingos.
Mining time-changing data streams.
In *Proc. KDD*, pages 97–106, 2001.

[66] Joao Gama, Ricardo Rocha, and Pedro Medas.
Accurate decision trees for mining high-speed data streams.
In *Proc. KDD*, pages 523–528, 2003.

[67] Isvani Frias-Blanco, Jose del Campo-Avila, Gonzalo Ramos-Jimenez, Andre CPLF Carvalho, Agustin Ortiz-Diaz, and Rafael Morales-Bueno.
Online adaptive decision trees based on concentration inequalities.
Knowl.-Based Syst., 104:179–194, 2016.

[68] Ludmila I Kuncheva.
Classifier ensembles for changing environments.
In *Int. Workshop Mult. Classifier Syst.*, pages 1–15, Cagliari, Italy, June 9-11, 2004. Springer.

[69] Alexey Tsymbal, Mykola Pechenizkiy, Pádraig Cunningham, and Seppo Puuronen.
Dynamic integration of classifiers for handling concept drift.
Inf. Fusion, 9(1):56–68, 2008.

[70] W Nick Street and YongSeog Kim.
A streaming ensemble algorithm (sea) for large-scale classification.
In *Proc. KDD*, pages 377–382, San Francisco, CA, USA, August 26-29, 2001.

[71] Nikunj C Oza and Stuart Russell.
Experimental comparisons of online and batch versions of bagging and boosting.
In *Proc. KDD*, pages 359–364, 2001.

[72] Albert Bifet, Geoff Holmes, and Bernhard Pfahringer.
Leveraging bagging for evolving data streams.
In *Proc. ECMLPKDD*, pages 135–150. Springer, 2010.

[73] J Zico Kolter and Marcus A Maloof.
Dynamic weighted majority: An ensemble method for drifting concepts.
J. Mach. Learn. Res., 8:2755–2790, 2007.

[74] Nick Littlestone and Manfred K Warmuth.
The weighted majority algorithm.
Inf. Comput., 108(2):212–261, 1994.

[75] Xu-Cheng Yin, Kaizhu Huang, and Hong-Wei Hao.
De2: Dynamic ensemble of ensembles for learning nonstationary data.
Neurocomputing, 165:14–22, 2015.

[76] Peng Zhang, Jun Li, Peng Wang, Byron J Gao, Xingquan Zhu, and Li Guo.
Enabling fast prediction for ensemble models on data streams.
In *Proc. KDD*, pages 177–185, San Diego, CA, USA, August 21-24, 2011.

[77] Lena Pietruczuk, Leszek Rutkowski, Maciej Jaworski, and Piotr Duda.
A method for automatic adjustment of ensemble size in stream data mining.
In *Proc. IJCNN*, pages 9–15, Anchorage, AK, USA, May 14-19, 2016. IEEE.

[78] Sheng-Chi You and Hsuan-Tien Lin.
A simple unlearning framework for online learning under concept drifts.
In *Pacific-Asia Conf. Knowl. Discov. Data Min.*, pages 115–126, Auckland, New Zealand, April 19-22, 2016. Springer.

[79] Yiliao Song, Guangquan Zhang, Haiyan Lu, and Jie Lu.
A fuzzy drift correlation matrix for multiple data stream regression.
In *Proc. FUZZ-IEEE*, pages 1–6. IEEE, 2020.

[80] Wei Duan, Junyu Xuan, Maoying Qiao, and Jie Lu.
Learning from the dark: Boosting graph convolutional neural networks with diverse negative samples.
In *Proc. AAAI*, pages 6550–6558, 2022.

[81] Ming Li, Alessio Micheli, Yu Guang Wang, Shirui Pan, Pietro Lió, Giorgio Stefano Gnecco, and Marcello Sanguineti.
Guest editorial: deep neural networks for graphs: theory, models, algorithms, and applications.
IEEE Trans. Neural Netw. & Learn. Syst., 35(4):4367–4372, 2024.

[82] Wei Duan, Junyu Xuan, Maoying Qiao, and Jie Lu.
Graph convolutional neural networks with diverse negative samples via decomposed determinant point processes.
IEEE Trans. Neural Netw. & Learn. Syst., 2023.

[83] Jianfei Li, Ruigang Zheng, Han Feng, Ming Li, and Xiaosheng Zhuang.
Permutation equivariant graph framelets for heterophilous graph learning.
IEEE Trans. Neural Netw. & Learn. Syst., 35(9):11634–11648, 2024.

[84] Weiwei Jiang and Jiayun Luo.

Graph neural network for traffic forecasting: A survey.
Expert Systems with Applications, 207:117921, 2022.

[85] Lu Bai, Lixin Cui, Yue Wang, Ming Li, Jing Li, Philip S. Yu, and Edwin R. Hancock.
Haqjsk: Hierarchical-aligned quantum jensen-shannon kernels for graph classification.
IEEE Trans. Neural Netw. & Learn. Syst., 36(11):6370–6384, 2024.

[86] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu.
Diffusion convolutional recurrent neural network: Data-driven traffic forecasting.
In *Proc. ICLR*, 2018.

[87] Bing Yu, Haoteng Yin, and Zhanxing Zhu.
Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting.
In *Proc. IJCNN*, pages 3634–3640, 2018.

[88] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang.
Graph wavenet for deep spatial-temporal graph modeling.
In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 1907–1913, 2019.

[89] Mengzhang Li and Zhanxing Zhu.
Spatial-temporal fusion graph neural networks for traffic flow forecasting.
In *Proc. AAAI*, volume 35, pages 4189–4196, 2021.

[90] Kan Guo, Yongli Hu, Zhen Qian, Yanfeng Sun, Junbin Gao, and Baocai Yin.
Dynamic graph convolution network for traffic forecasting based on latent network of laplace matrix estimation.
IEEE Trans. Intell. Transp. Syst., 23(2):1009–1018, 2022.

[91] Fuxian Li, Jie Feng, Huan Yan, Guangyin Jin, Fan Yang, Funing Sun, Depeng Jin, and Yong Li.
Dynamic graph convolutional recurrent network for traffic prediction: Benchmark and solution.
ACM Trans. Knowl. Discov. Data, 2023.

[92] Chuanpan Zheng, Xiaoliang Fan, Cheng Wang, and Jianzhong Qi.
Gman: A graph multi-attention network for traffic prediction.
In *Proc. AAAI*, volume 34, pages 1234–1241, 2020.

[93] Zezhi Shao, Zhao Zhang, Fei Wang, and Yongjun Xu.
Pre-training enhanced spatial-temporal graph neural network for multivariate time series forecasting.
In *Proc. KDD*, pages 1567–1577, 2022.

[94] Zezhi Shao, Zhao Zhang, Wei Wei, Fei Wang, Yongjun Xu, Xin Cao, and Christian S. Jensen.
Decoupled dynamic spatial-temporal graph neural network for traffic forecasting.
Proc. VLDB Endow., 15(11):2733,Äì2746, 2022.

[95] Lei Bai, Lina Yao, Can Li, Xianzhi Wang, and Can Wang.
Adaptive graph convolutional recurrent network for traffic forecasting.
Proc. NeurIPS, 33:17804–17815, 2020.

[96] Chao Shang, Jie Chen, and Jinbo Bi.
Discrete graph structure learning for forecasting multiple time series.
Proc. ICLR, 2021.

[97] Yiliao Song, Jie Lu, Haiyan Lu, and Guangquan Zhang.
Fuzzy clustering-based adaptive regression for drifting data streams.

IEEE Trans. Fuzzy Syst., 28(3):544–557, 2019.

[98] Michael Bonnell Harries, Claude Sammut, and Kim Horn.
Extracting hidden context.
Mach. Learn., 32(2):101–126, 1998.

[99] Geoffrey I Webb, Roy Hyde, Hong Cao, Hai Long Nguyen, and Francois Petitjean.
Characterizing concept drift.
Data Min. Knowl. Discov., 30(4):964–994, 2016.

[100] Indrė Žliobaitė, Mykola Pechenizkiy, and Joao Gama.
An overview of concept drift applications.
Big Data Anal., pages 91–114, 2016.

[101] Anjin Liu, Jie Lu, and Guangquan Zhang.
Diverse instance-weighting ensemble based on region drift disagreement for concept drift adaptation.
IEEE Trans. Neural Netw. & Learn. Syst., 32(1):293–307, 2020.

[102] Yongchan Kwon, Joong-Ho Won, Beom Joon Kim, and Myunghee Cho Paik.
Uncertainty quantification using bayesian neural networks in classification: Application to biomedical image segmentation.
Comput. Stat. & Data Anal., 142:106816, 2020.

[103] Aaron Klein, Stefan Falkner, Jost Tobias Springenberg, and Frank Hutter.
Learning curve prediction with bayesian neural networks.
In *Proc. ICLR*, 2017.

[104] Bin Wang, Jie Lu, Zheng Yan, Huaishao Luo, Tianrui Li, Yu Zheng, and Guangquan Zhang.

Deep uncertainty quantification: A machine learning approach for weather forecasting.

In *Proc. KDD*, pages 2087–2095, 2019.

[105] Bach Tran, Anh Duc Nguyen, Linh Ngo Van, and Khoat Than.

Dynamic transformation of prior knowledge into bayesian models for data streams.

IEEE Trans. Knowl. Data Eng., 2021.

[106] Yang Gao, Swarup Chandra, Yifan Li, Latifur Khan, and Thuraisingham Bhavani.

Saccos: A semi-supervised framework for emerging class detection and concept drift adaption over data streams.

IEEE Trans. Knowl. Data Eng., 34(3):1416–1426, 2022.

[107] Meenal Jain, Gagandeep Kaur, and Vikas Saxena.

A k-means clustering and svm based hybrid concept drift detection technique for network anomaly detection.

Expert Syst. Appl., 193:116510, 2022.

[108] Weike Liu, Hang Zhang, Zhaoyun Ding, Qingbao Liu, and Cheng Zhu.

A comprehensive active learning method for multiclass imbalanced data streams with concept drift.

Knowl.-Based Syst., 215:106778, 2021.

[109] Yu Sun, Ke Tang, Leandro L Minku, Shuo Wang, and Xin Yao.

Online ensemble learning of data streams with gradually evolved classes.

IEEE Trans. Knowl. Data Eng., 28(6):1532–1545, 2016.

[110] Paola Castro-Cabrera, G Castellanos-Dominguez, Carlos Mera, Luis Franco-Marín, and Mauricio Orozco-Alzate.

Adaptive classification using incremental learning for seismic-volcanic signals with concept drift.

J. Volcanol. Geotherm. Res., 413:107211, 2021.

[111] Ming Zhou, Yiliao Song, Guangquan Zhang, Bin Zhang, and Jie Lu.
An efficient bayesian neural network for multiple data streams.
In *Proc. IJCNN*, pages 1–8. IEEE, 2021.

[112] Ahsanul Haque, Hemeng Tao, Swarup Chandra, Jie Liu, and Latifur Khan.
A framework for multistream regression with direct density ratio estimation.
In *Proc. AAAI*, volume 32, 2018.

[113] Li Yang and Abdallah Shami.
A lightweight concept drift detection and adaptation framework for iot data streams.
IEEE Internet Things Mag., 4(2):96–101, 2021.

[114] Bo Dong, Yifan Li, Yang Gao, Ahsanul Haque, Latifur Khan, and Mohammad M Masud.
Multistream regression with asynchronous concept drift detection.
In *IEEE Int. Conf. Big Data*, pages 596–605. IEEE, 2017.

[115] Swarup Chandra, Ahsanul Haque, Latifur Khan, and Charu Aggarwal.
An adaptive framework for multistream classification.
In *Proc. CIKM*, pages 1181–1190, 2016.

[116] Yi-Fan Li, Yang Gao, Gbadebo Ayoade, Hemeng Tao, Latifur Khan, and Bhavani Thuraisingham.
Multistream classification for cyber threat data with heterogeneous feature space.
In *Proc. WWW*, page 2992,Äì2998. Association for Computing Machinery, 2019.

[117] En Yu, Yiliao Song, Guangquan Zhang, and Jie Lu.
Learn-to-adapt: Concept drift adaptation for hybrid multiple streams.

Neurocomputing, 496:121–130, 2022.

[118] Shengnan Guo, Youfang Lin, Huaiyu Wan, Xiucheng Li, and Gao Cong.
Learning dynamics and heterogeneity of spatial-temporal graph data for traffic forecasting.
IEEE Trans. Knowl. Data Eng., 2021.

[119] Eric Jang, Shixiang Gu, and Ben Poole.
Categorical reparameterization with gumbel-softmax.
In *Proc. ICLR*, 2017.

[120] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh.
The concrete distribution: A continuous relaxation of discrete random variables.
In *Proc. ICLR*, 2017.

[121] Ilya Sutskever, Oriol Vinyals, and Quoc V Le.
Sequence to sequence learning with neural networks.
In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Proc. NeurIPS*, volume 27, 2014.

[122] Diederik P Kingma and Jimmy Ba.
Adam: A method for stochastic optimization.
arXiv preprint arXiv:1412.6980, 2014.

[123] Hosagrahar V Jagadish, Johannes Gehrke, Alexandros Labrinidis, Yannis Papakonstantinou, Jignesh M Patel, Raghu Ramakrishnan, and Cyrus Shahabi.
Big data and its technical challenges.
Communications of the ACM, 57(7):86–94, 2014.

[124] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun.

Graph neural networks: A review of methods and applications.

AI Open, 1:57–81, 2020.

[125] Minghuan Liu.

Ai-challenger-weather-forecast, 2019.

[126] Supriya Agrahari and Anil Kumar Singh.

Concept drift detection in data stream mining: A literature review.

J. King Saud Univ. - Comput. Inf. Sci., 34(10):9523–9540, 2022.

[127] Rodrigo G. F. Soares and Leandro L. Minku.

Osnn: An online semisupervised neural network for nonstationary data streams.

IEEE Trans. Neural Netw. & Learn. Syst., pages 1–13, 2021.

[128] Ahmad Abbasi, Abdul Rehman Javed, Chinmay Chakraborty, Jamel Nebhen, Wisha Zehra, and Zunera Jalil.

Elstream: An ensemble learning approach for concept drift detection in dynamic social big data stream learning.

IEEE Access, 9:66408–66419, 2021.

[129] Denise Maria Vecino Sato, Sheila Cristiana De Freitas, Jean Paul Barddal, and Edson Emilio Scalabrin.

A survey on concept drift in process mining.

ACM Comput. Surv., 54(9):1–38, 2021.

[130] Hang Yu, Jie Lu, Anjin Liu, Bin Wang, Ruimin Li, and Guangquan Zhang.

Real-time prediction system of train carriage load based on multi-stream fuzzy learning.

IEEE Trans. Intell. Transp. Syst., 23(9):15155–15165, 2022.

[131] JQ James.

Graph construction for traffic prediction: a data-driven approach.
IEEE Trans. Intell. Transp. Syst., 23(9):15015–15027, 2022.

[132] Ming Zhou, Jie Lu, Yiliao Song, and Guangquan Zhang.
Multi-stream concept drift self-adaptation using graph neural network.
IEEE Trans. Knowl. Data Eng., 35(12):12828–12841, 2023.

[133] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio.
Empirical evaluation of gated recurrent neural networks on sequence modeling.
Proc. NeurIPS, 2014.

[134] Suresh Chavhan and Pallapa Venkataram.
Prediction based traffic management in a metropolitan area.
J. Traffic Transp. Eng. (Engl. Ed.), 7(4):447–466, 2020.

[135] Tian Wang, Jiahui Chen, Jinhu Lü, Kexin Liu, Aichun Zhu, Hichem Snoussi, and Baochang Zhang.
Synchronous spatiotemporal graph transformer: A new framework for traffic data prediction.
IEEE Trans. Neural Netw. & Learn. Syst., 2022.

[136] Kun Wang, Jie Lu, Anjin Liu, Guangquan Zhang, and Li Xiong.
Evolving gradient boost: A pruning scheme based on loss improvement ratio for learning under concept drift.
IEEE Trans. Cybern., 53(4):2110–2123, 2021.

[137] Liheng Yuan, Heng Li, Beihao Xia, Cuiying Gao, Mingyue Liu, Wei Yuan, and Xinge You.
Recent advances in concept drift adaptation methods for deep learning.
In *Proc. IJCNN*, pages 5654–5661. Austria: International Joint Conferences on Artificial Intelligence Organization, 2022.

[138] Hang Yu, Weixu Liu, Jie Lu, Yimin Wen, Xiangfeng Luo, and Guangquan Zhang.
Detecting group concept drift from multiple data streams.
Pattern Recognition, 134:109113, 2023.

[139] Saeed Rahmani, Asiye Baghbani, Nizar Bouguila, and Zachary Patterson.
Graph neural networks for intelligent transportation systems: A survey.
IEEE Trans. Intell. Transp. Syst., 2023.

[140] Ming Zhou, Jie Lu, Pengqian Lu, and Guangquan Zhang.
Dynamic graph regularization for multi-stream concept drift self-adaptation.
IEEE Trans. Knowl. Data Eng., 36(11):6016–6028, 2024.

[141] Indrė Žliobaitė.
Learning under concept drift: an overview.
arXiv preprint arXiv:1010.4784, 2010.

[142] Ben Halstead, Yun Sing Koh, Patricia Riddle, Mykola Pechenizkiy, and Albert Bifet.
Fall: A modular adaptive learning platform for streaming data.
In *Proc. ICDE*, pages 3619–3622. IEEE, 2023.

[143] Ahsanul Haque, Zhuoyi Wang, Swarup Chandra, Bo Dong, Latifur Khan, and Kevin W Hamlen.
Fusion: An online method for multistream classification.
In *Proc. CIKM*, pages 919–928, 2017.

[144] Xie Renchunzi and Mahardhika Pratama.
Automatic online multi-source domain adaptation.
Information Sciences, 582:480–494, 2022.