Aalborg Universitet

AALBORG UNIVERSITY

Aspects of Deep Spatio-Temporal Analytics for Time Series, Streaming, and Trajectory Data

Miao, Hao

DOI (link to publication from Publisher):
10.54337/aau757771238

Publication date:
2024

Document Version
Publisher's PDF, also known as Version of record

Link to publication from Aalborg University

Citation for published version (APA):
Miao, H. (2024). *Aspects of Deep Spatio-Temporal Analytics for Time Series, Streaming, and Trajectory Data*. Aalborg University Open Publishing. https://doi.org/10.54337/aau757771238

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal -

Take down policy
If you believe that this document breaches copyright please contact us at vbn@aub.aau.dk providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from vbn.aau.dk on: March 06, 2026

ASPECTS OF DEEP SPATIO-TEMPORAL ANALYTICS FOR TIME SERIES, STREAMING, AND TRAJECTORY DATA

BY
HAO MIAO

PhD Thesis 2024



AALBORG UNIVERSITY
DENMARK

Aspects of Deep Spatio-Temporal Analytics for Time Series, Streaming, and Trajectory Data

Ph.D. Dissertation

Hao Miao

PhD Thesis 2024

Submitted: October 2024

Main Supervisor: Professor Christian S. Jensen
Aalborg University

Co-supervisors: Professor Chenjuan Guo
East China Normal University

Associate Professor Yan Zhao
Aalborg University

Assessment: Associate Professor Peter Dolog (chair)
Aaborg University, Denmark

Professor Ira Assent
Aarhus University, Denmark

Distinguished Professor Themistoklis Palpanas
Université Paris Cité, France

PhD Series: Technical Faculty of IT and Design, Aalborg University

Department: Department of Computer Science

ISSN: 2446-1628
ISBN: 978-87-85239-35-8

Published by:
Aalborg University Open Publishing
Kroghstræde 1-3
DK – 9220 Aalborg Øst
aauopen@aau.dk

© Copyright: Hao Miao

Abstract

The widespread deployment of wireless and mobile devices results in a proliferation of spatio-temporal data (e.g., time series, streaming, and trajectory data) that is essential for applications, e.g., traffic and air quality prediction, where spatio-temporal analytics plays a key role in ensuring safety, predictability, and reliability. While recent advances in deep learning have demonstrated superior performance in spatio-temporal analytics, many approaches struggle to adapt to real-world conditions.

In particular, existing methods suffer from three limitations: 1) deep learning methods typically require large-scale training data, incurring high storage and computational costs; 2) when applied to streaming data, many models suffer from catastrophic forgetting, resulting in deteriorating prediction accuracy over time; and 3) existing solutions often assume centralized data, which leads to privacy concerns and fails to exploit decentralized data processing.

This Ph.D. study aims to systematically study spatio-temporal analytics using emerging deep learning techniques. Specifically, we target four types of functionality: time series dataset condensation, streaming spatio-temporal prediction, federated trajectory recovery, and federated trajectory similarity learning.

First, we address the problem of time series dataset condensation. The goal is to reduce training costs by summarizing large datasets into smaller, synthetic datasets that can then be used for training instead. We introduce TimeDC, an efficient time series dataset condensation framework that uses two-fold modal matching. TimeDC encompasses a time series feature extraction module for effective feature learning, a decomposition-driven frequency matching module for achieving temporal correlation preservation, and a curriculum-based trajectory matching module for ensuring that the synthetic datasets capture key patterns in the original dataset. Extensive experiments on real data verify the effectiveness of TimeDC.

Second, we investigate spatio-temporal prediction on streaming data. We propose URCL, a unified replay-based continuous learning framework with three key modules: data integration, spatio-temporal continuous representation learning, and spatio-temporal prediction. Specifically, a spatio-temporal

mixup mechanism is introduced to alleviate catastrophic forgetting, and a simple siamese network is designed to facilitate holistic feature learning. Extensive experiments on real datasets offer evidence of the effectiveness of URCL in continuous learning environments.

Third, we study the problem of federated trajectory recovery, focusing on privacy preservation and enabling decentralized training. We propose LighTR+, a horizontal federated framework, which consists of a lightweight local trajectory embedding module, an intra-domain knowledge distillation module, and a meta-knowledge enhanced local-global training scheme. LighTR+ alleviates intra- and inter-domain interferences across clients while reducing communication costs between clients and the server, thereby facilitating privacy protection and improving efficiency.

Fourth, we explore federated trajectory similarity learning for decentralized data processing. We propose the F-TSL framework based on horizontal federated learning, a server-clients architecture. The framework includes a local trajectory preprocessing and learning module for clients, a privacy-preserving clustering module, and a server-side aggregation module, where the privacy-preserving clustering module leverages differential privacy to handle data heterogeneity across clients. Experiments on real data demonstrate the effectiveness of F-TSL at preserving privacy and facilitating efficient decentralized learning.

Resumé

Den udbredte anvendelse af trådløse og mobile enheder resulterer i en overflod af spatio-temporale data (data med referencer til tid og sted; fx tidsserier og datastrømme), som er afgørende for applikationer så som forudsigelse af trafik og luftkvalitet. Selvom de seneste fremskridt inden for neurale netværk har resulteret i metoder med overlegen performance inden for analyse af spatio-temporale data, så er er der stadig udfordringer, når metoderne skal tilpasses praktiske forhold.

Eksisterende metoder udviser ofte tre begrænsninger: 1) metoder baseret på neurale netværk kræver typisk store mængder træningsdata, hvilket medfører høje lagrings- og beregningsomkostninger; 2) når de anvendes på datastrømme, lider mange metoder af såkaldt katastrofal glemsel, hvilket over tid forringer deres evne til at lave nøjagtige forudsigelser; og 3) eksisterende metoder antager ofte at data er centraliserede, hvilket kan give udfordringer med at sikre privathed og med at udnytte muligheder for udføre decentraliserede beregninger.

Dette ph.d.-studie har til formål at opfinde metoder til analyse af spatio-temporale ved hjælp af nye teknikker inden for neurale netværk. Specifikt fokuserer studiet på fire typer funktionalitet: kondensering af tidsseriedatasæt, prædiktion baseret på spatio-temporale datastrømme, distribueret rekonstruktion af spatio-temporale bevægelsesdata og distribueret læring af similaritet af bevægelsesdata.

Først adresserer vi problemet med kondensering af tidsseriedatasæt. Her reducerer vi omkostningerne forbundet med at træne neurale netværksmodeller ved at kondensere store træningsdatasæt til mindre, syntetiske datasæt. Til dette formål introducerer vi TimeDC, som omfatter en modul til effektivt at repræsentere tidserier som vektorer samt to moduler, der har til formål at bevare henholdsvis korrelationer og mønstre fra det oprindelige datasæt i det kondenserede datasæt. Omfattende eksperimenter på virkelige data viser, at TimeDC fungere efter hensigten.

Dernæst beskriver vi en metode, kaldet URCL, der muliggør forudsigelser baseret på spatio-temporale datastrømme. URCL har tre nøglemoduler: dataintegration, kontinuerlig læring af vektorrepræsentationer af spatio-temporale

datastrømme samt spatio-temporal forudsigelse. En såkaldt mixup-mekaniske sikrer mod katastrofal glemsel og et simpelt siamesisk neuralt netværk muliggør holistisk indlæring af vektorrepræsentationer. Omfattende eksperimenter på virkelige datasæt bekræfter, at URCL er velfungerende.

For det tredje præsenterer vi en metode, kaldet LighTR+, til distribueret rekonstruktion af spatio-temporale bevægelsesdata og muliggør dermed sikring af privathed og udnyttelse af decentrale beregningsresurser. LighTR+ omfatter et modul, der muliggør decentral repræsentation af bevægelsesdata som vektorer, et vidensdistillationsmodul, og et modul, der muliggør samarbejde om modeltræning mellem de lokale klienter og den centrale server i LighTR+. Ved hjælp af disse moduler muliggør LighTR+ både privathed og forbedret effektivitet.

Endelig præsenterer vi en metode, F-TSL, der muliggør distribueret læring af similaritet af spatio-temporale bevægelsesdata. F-TSL omfatter et modul til klient-baseret præprocessering og læring. Et andet modul benytter såkaldt differentiel privathed til at muliggøre privathedsbevarende gruppering af bevægelsesdata baseret på similaritet. Endelig rummer F-TSL et server-baseret modul, der muliggør aggregering af data på tværs af klienter. Eksperimenter på virkelige data giver empirisk evidens for, at F-TSL Er i stand til at bevare privathed samt at facilitere effektiv distribueret læring.

Acknowledgements

I would like to thank many people who have supported me during my Ph.D. studies.

First and foremost, I would like to express my sincere gratitude to my supervisor Prof. Christian S. Jensen for his continuous support of my Ph.D. study and research. I have gained a wealth of knowledge and skills, covering topics such as scientific writing and research methods, due to his meticulous guidance. Prof. Christian S. Jensen has always been encouraging and open to various research topics, which granted me freedom to explore different research fields. This experience has equipped me with a broad research vision that will benefit me in my future academic career. It has been my pleasure to be one of his Ph.D. students. I deeply appreciate his invaluable guidance and instructions throughout my Ph.D. study.

Second, I would like to express my appreciation to my co-supervisor Prof. Chenjuan Guo. I appreciate that she offered me the opportunity to be one of her Ph.D. students. I would like to thank Prof. Chenjuan Guo for her unique ideas and valuable comments in preparing and revising my papers, which taught me how to write good research papers. Additionally, her guidance in my daily life has helped me adapt to life abroad.

Third, I am especially grateful to my co-supervisor Assoc. Prof. Yan Zhao. I have learned a lot from her, from how to formulate a research problem to how to write good research papers. I appreciate the considerable time she dedicated to revising my drafts. I will always remember the scene of Yan working late into the night in her office to help me revise my first paper. Without her strong help, I would not have been able to complete this challenging journey and have these publications.

I would like to thank Prof. Gao Cong, who hosted me as a visiting Ph.D. student at Nanyang Technological University. I appreciate his support and insightful comments on my research.

In addition, I would like to thank all my colleagues at the Data Engineering, Science and Systems group at Aalborg University (AAU) for a fun-filled and friendly working environment. I would especially like to thank Prof. Bin Yang for his constructive and innovative suggestions in helping me revise my papers

during the past three years. I also thank my colleagues Xinle Wu, Dazhuo Qiu, and Miao Zhang for not making me feel alone. In addition, I thank the administrative staff at AAU, especially Helle Westmark, Helle Schroll, and Susanne Taunsig Larsen who have made my Ph.D. life easier. Further, I would like to thank my co-authors, especially Ziqiao Liu, Ronghui Xu, and Chenxi Liu, for their substantial support in terms of coding and writing. It has been a pleasure to work with them.

Finally, I would like to thank my family, especially my parents, who always stand by me and constantly encourage me. I am thankful to Chenxi Liu, who plays a positive role during my Ph.D. period.

Contents

Abstract iii

Resumé v

Acknowledgements vii

Thesis Details xiii

I Thesis Summary 1

Introduction 3
1 Background and Motivation 3
2 Thesis Structure 5

1 Less is More: Efficient Time Series Dataset Condensation via Two-fold Modal Matching 7
1 Problem Motivation and Statement 7
2 Preliminaries 9
2.1 Basic Concepts 9
2.2 Problem Definition 9
3 Methodology 10
3.1 Time Series Feature Extraction 11
3.2 Decomposition-Driven Frequency Matching 11
3.3 Curriculum Training Trajectory Matching 12
3.4 Overall Objective Function 14
4 Experimental Evaluation 15
4.1 Experimental Setup 15
4.2 Experimental Results 16

2 A unified replay-based continuous learning framework for spatio-temporal prediction on streaming data 21
1 Problem Motivation and Statement 21

2 Preliminaries . . . 23
2.1 Basic Concepts . . . 23
2.2 Problem Definition . . . 23
3 Methodology . . . 24
3.1 Data Integration . . . 24
3.2 Spatio-temporal Continuous Representation Learning . . 26
3.3 Spatio-Temporal Prediction . . . 30
3.4 Overall Objective Function . . . 32
4 Experimental Evaluation . . . 33
4.1 Experimental Setup . . . 33
4.2 Experimental Results . . . 33

3 LightTR+: A Lightweight Incremental Framework for Federated Trajectory Recovery 39
1 Problem Motivation and Statement . . . 39
2 Preliminaries . . . 42
2.1 Basic Concepts . . . 42
2.2 Problem Definition . . . 43
3 Methodology . . . 44
3.1 Local Trajectory Preprocessing and Light Embedding . . 44
3.2 Meta-knowledge Enhanced Local-Global Training 48
4 Experimental Evaluation . . . 52
4.1 Experimental Setup . . . 52
4.2 Experimental Results . . . 54

4 Federated Trajectory Similarity Learning with Privacy-Preserving Clustering 59
1 Problem Motivation and Statement . . . 59
2 Preliminaries . . . 61
2.1 Basic Concepts . . . 61
2.2 Problem Definition . . . 62
3 Methodology . . . 63
3.1 Local Trajectory Preprocessing and Learning . . . 63
3.2 Privacy-Preserving Clustering . . . 65
3.3 Server-Side Aggregation . . . 68
4 Experimental Evaluation . . . 71
4.1 Experimental Setup . . . 71
4.2 Experimental Results . . . 72

5 Conclusion and Future Work 77
1 Conclusion . . . 77
2 Future Work . . . 78
References . . . 78

II Papers **85**

A Less is More: Efficient Time Series Dataset Condensation via Two-fold Modal Matching **87**
1 Introduction . . . 89
2 Preliminaries . . . 93
2.1 Time Series Seasonal-Trend Decomposition via Moving Average . . . 93
2.2 Dataset Condensation over Time Series . . . 94
3 Methodology . . . 95
3.1 Framework Overview . . . 95
3.2 Time Series Feature Extraction . . . 97
3.3 Decomposition-Driven Frequency Matching . . . 99
3.4 Curriculum Training Trajectory Matching . . . 100
3.5 Overall Objective Function . . . 103
4 Experimental Evaluation . . . 104
4.1 Experimental Setup . . . 104
4.2 Experimental Results . . . 106
4.3 Application on Streaming Time Series . . . 112
4.4 Performance on Time Series Classification . . . 114
5 Related work . . . 114
5.1 Time Series Modeling . . . 114
5.2 Coreset and Dataset Condensation . . . 115
6 Conclusion . . . 115
References . . . 116

B A Unified Replay-based Continuous Learning Framework for Spatio-Temporal Prediction on Streaming Data **121**
1 Introduction . . . 123
2 Related work . . . 126
2.1 Spatio-Temporal Data Prediction . . . 126
2.2 Continuous Learning . . . 127
3 Problem Statement . . . 127
4 Methodology . . . 128
4.1 Framework Overview . . . 128
4.2 Data Integration . . . 129
4.3 Spatio-temporal Continuous Representation Learning . . . 131
4.4 Spatio-Temporal Prediction . . . 135
4.5 Overall Objective Function . . . 139
5 Experimental Evaluation . . . 140
5.1 Experimental Setup . . . 140
5.2 Experimental Results . . . 143
6 Conclusion . . . 148

References . . . 149

C LightTR+: A Lightweight Incremental Framework for Federated Trajectory Recovery 155
1 Introduction . . . 157
2 Preliminaries . . . 161
3 Analysis and Observations . . . 163
4 Methodology . . . 165
4.1 Framework Overview . . . 166
4.2 Local Trajectory Preprocessing and Light Embedding . . . 166
4.3 Meta-knowledge Enhanced Local-Global Training . . . 170
5 Experimental Study . . . 175
5.1 Setup . . . 175
5.2 Experimental Results . . . 177
6 Related Work . . . 183
6.1 Trajectory Recovery . . . 183
6.2 Federated Learning . . . 184
6.3 Incremental Learning . . . 185
7 Conclusion . . . 185
References . . . 185

D Federated Trajectory Similarity Learning with Privacy-Preserving Clustering 191
1 Introduction . . . 193
2 Preliminaries . . . 196
3 Discussion . . . 198
4 Methodology . . . 200
4.1 Framework Overview . . . 200
4.2 Local Trajectory Preprocessing and Learning . . . 200
4.3 Privacy-Preserving Clustering . . . 203
4.4 Server-Side Aggregation . . . 206
5 Experimental Evaluation . . . 208
5.1 Experimental Setup . . . 208
5.2 Experimental Results . . . 210
6 Related Work . . . 218
6.1 Trajectory Similarity . . . 218
6.2 Federated Learning . . . 219
7 Conclusion . . . 219
References . . . 220

Thesis Details

Thesis Title:	Aspects of Deep Spatio-Temporal Analytics for Time Series, Streaming, and Trajectory Data
Ph.D. Student:	Hao Miao
Supervisors:	Prof. Christian S. Jensen, Aalborg University Prof. Chenjuan Guo, East Normal China University Assoc. Prof. Yan Zhao, Aalborg University

The main body of this thesis consists of the following papers.

[A] **Hao Miao**, Ziqiao Liu, Yan Zhao, Chenjuan Guo, Bin Yang, Kai Zheng, Christian S. Jensen, "*Less is More: Efficient Time Series Dataset Condensation via Two-fold Modal Matching*," *in PVLDB*, 2025.

[B] **Hao Miao**, Yan Zhao, Chenjuan Guo, Bin Yang, Kai Zheng, Feiteng Huang, Jiandong Xie, Christian S. Jensen, "*A Unified Replay-based Continuous Learning Framework for Spatio-Temporal Prediction on Streaming Data*," *in ICDE*, pp. 1050-1062, 2024.

[C] **Hao Miao**, Ziqiao Liu, Yan Zhao, Chenxi Liu, Chenjuan Guo, Bin Yang, Kai Zheng, Huan Li, Christian S. Jensen, "*LightTR+: A Lightweight Incremental Framework for Federated Trajectory Recovery*," *TKDE (in submission)*, 2024.

[D] **Hao Miao**, Ziqiao Liu, Yan Zhao, Kai Zheng, Yupu Zhang, Christian S. Jensen, "*Federated Trajectory Similarity Learning with Privacy-Preserving Clustering*," *in ICDE (under minor revision)*, 2024.

In addition to the above papers, I have coauthored the following five papers as part of my Ph.D. studies, which are not included in the thesis.

[1] **Hao Miao**, Jiaxing Shen, Jiannong Cao, Jiangnan Xia, Senzhang Wang, "*MBA-STNet: Bayes-enhanced discriminative multi-task learning for flow prediction*," *TKDE*, vol. 35, no. 7, pp. 7164–7177, 2023.

[2] **Hao Miao**, Xiaolong Zhong, Jiaxin Liu, Yan Zhao, Xiangyu Zhao, Weizhu Qian, Kai Zheng, Christian S. Jensen, "*Task Assignment With Efficient Federated Preference Learning in Spatial Crowdsourcing*," *TKDE*, vol. 36, no. 4, pp. 1800–1814, 2024.

[3] Ziqiao Liu, **Hao Miao**, Yan Zhao, Chenxi Liu, Kai Zheng, Huan Li, "*LightTR: A Lightweight Framework for Federated Trajectory Recovery*," *in ICDE*, pp. 4422-4434, 2024.

[4] Ronghui Xu, **Hao Miao**, Senzhang Wang, Philip S. Yu, Jianxin Wang, "*PeFAD: A Parameter-Efficient Federated Framework for Time Series Anomaly Detection*," *in SIGKDD*, pp. 3621-3632, 2024.

[5] **Hao Miao**, Yan Zhao, Chenjuan Guo, Bin Yang, Kai Zheng, Christian S. Jensen, "*Spatio-Temporal Prediction on Streaming Data: A Unified Federated Continuous Learning Framework*," *TKDE (under major revision)*, 2024.

This thesis has been submitted for assessment in partial fulfillment of the PhD degree. The thesis is based on the submitted or published scientific papers which are listed above. Parts of the papers are used directly or indirectly in the extended summary of the thesis. As part of the assessment, co-author statements have been made available to the assessment committee and are also available at the Faculty. The permission for using the published and accepted articles in the thesis have been obtained from the corresponding publishers with the condition that they are cited and copyrights are placed prominently in the references. In reference to IEEE copyrighted material, which is used with permission in this thesis, the IEEE does not endorse any of Aalborg University's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to `http://www.ieee.org/publications_standards/publications/rights/rights_link.html` to learn how to obtain a License from RightsLink.

Part I

Thesis Summary

Introduction

The thesis focuses on deep learning based spatio-temporal analytics on time series, streaming, and trajectory data. In this section, we first introduce the background and motivation, followed by the thesis structure.

1 Background and Motivation

With the proliferation of edge computing and mobile sensing, massive volumes of spatio-temporal data, such as time series and trajectories, comprising of millions of observations, are being recorded and stored into database management systems (DBMSs) [31]. Spatio-temporal data often refers to data that encompasses both spatial and temporal dimensions. For example, populations of in-road sensors provide data that captures traffic flow in multiple locations across time. As suggested in Figure 1, effective spatio-temporal analytics is critically important as it enables prominent real-world applications, such as urban planning [68], crime analysis [87], and smart transportation [47].

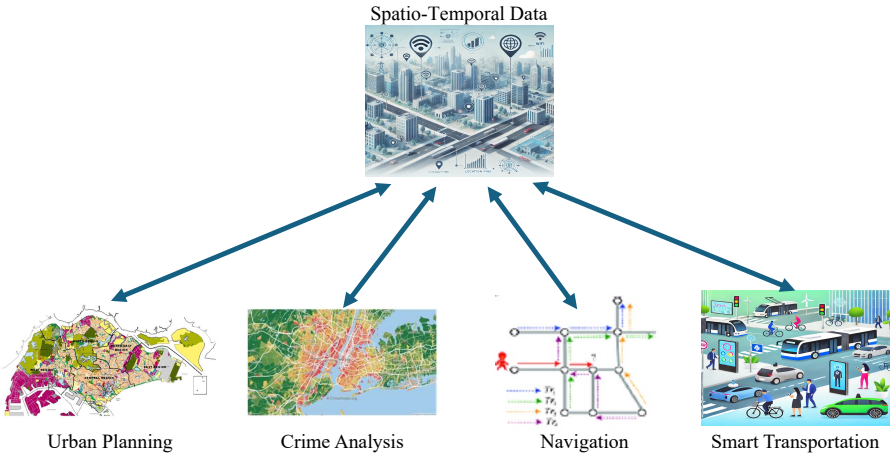


Fig. 1: Spatio-Temporal Data based Applications [36, 50]

Spatio-temporal data often takes different forms in real-world scenarios. In this study, we focus on spatio-temporal analytics for time series, streaming, and trajectory data, where a time series is a time ordered sequence of observations, streaming spatio-temporal data consists of continuously generated observations that evolve over time, and a trajectory is a sequence of GPS points at certain discrete timestamps that capture the movement of an object.

The key aspect of spatio-temporal analytics is to learn spatial and temporal correlations in the data effectively. Traditional methods are mostly based on statistical models [37, 58], e.g., ARIMA [32]. However, statistical models cannot capture complex spatial and temporal correlations effectively, due to their limited learning capacity. Recently, with the advancement of deep learning techniques, various deep learning based methods address spatio-temporal analytics [47, 73, 84]. These methods benefit from the massively available spatio-temporal data and the powerful feature extraction capacities of neural networks. Deep learning based methods, also known as deep spatio-temporal neural networks, outperform traditional statistical models when used for supporting a broad range of downstream tasks.

Despite the success of deep learning spatio-temporal analytics methods, they still face three challenges. First, most existing deep learning based methods are computationally intensive, requiring vast amounts of processing power and consuming considerable time. This leads to a bottleneck in real-time, resource-constrained applications. Second, it is challenging to enable streaming spatio-temporal learning that avoids catastrophic forgetting, which is the tendency to abruptly forget previously learned knowledge when learning new knowledge. Third, massive volumes of spatio-temporal data are being generated in a decentralized fashion with the widespread deployment of edge computing and devices. However, most existing methods assume centralized training data and fail to take into account the decentralized setting.

To address these shortcomings, we address the following four problems in the thesis.

1) **Time Series Dataset Condensation.** To reduce the cost associated with the use of large data for model, we propose an efficient time series dataset condensation framework, TimeDC, that employs two-fold modal matching. We design a time series feature extraction module with well-customized *TSOperators*. A curriculum training trajectory matching module is proposed to enable effective and generalized time series dataset condensation. To reduce the discrepancy between frequencies in the original and condensed time series, we propose a decomposition-driven frequency matching module.

2) **Continuous Spatio-Temporal Data Prediction.** To enable stream learning, we propose a unified replay-based continuous learning (URCL) framework for spatio-temporal prediction on streaming data. We pro-

pose a spatio-temporal mixup mechanism and a ranking-based maximally interfered retrieval sampling strategy to alleviate catastrophic forgetting. In addition, we propose a spatio-temporal continuous learning module to learn holistic features to enable accuracy. A common spatio-temporal prediction module is proposed by identifying the commonalities among existing methods.

3) **Federated Trajectory Recovery.** To develop a decentralized training framework, we study federated trajectory recovery, where a lightweight incremental framework, LighTR+, is proposed based on horizontal federated learning (i.e., a cleint-server architecture). Given the limited processing capabilities of edge devices, LightTR+ includes a lightweight local trajectory embedding module. To mitigate catastrophic forgetting, we propose an intra-domain knowledge distillation module. Additionally, LightTR+ features a meta-knowledge enhanced local-global training scheme, which reduces communication costs between the server and clients, further improving efficiency.

4) **Federated Trajectory Similarity Learning.** We propose F-TSL, a federated trajectory similarity learning framework, based on horizontal federated learning. We design a local trajectory preprocessing and learning module to learn spatio-temporal correlations based on a local trajectory learning model. In addition, to contend with trajectory heterogeneity across clients, we propose a privacy-preserving clustering module.

2 Thesis Structure

Figure 2 shows the thesis structure. Overall, the thesis is motivated by three challenges: high data storage and computation costs, streaming learning, and decentralized training and privacy. First, we study efficient time series dataset condensation aiming to reduce the cost associated with the use of large data, which is the topic of Paper A [45]. Next, we study streaming spatio-temporal prediction that avoids catastrophic forgetting, which is the topic of Paper B [48]. Finally, we study federated trajectory recovery and federated trajectory similarity learning to enable decentralized training and privacy preservation, which are the topics of Papers C [46] and D [44].

Chapter 1 presents an efficient time series dataset condensation framework. Key aspects of the proposal include: i) a time series feature extraction module for effective feature extraction; ii) a decomposition-driven frequency matching module to preserve temporal patterns in an original, large time series dataset in the condensed time series dataset; iii) a curriculum training trajectory matching module to imitate the training dynamics on the original dataset; and iv) extensive experiments to assess pertinent properties of the proposal.

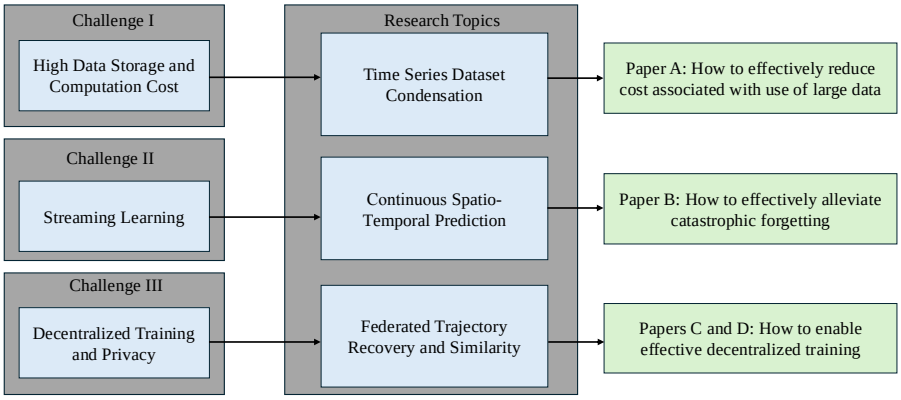


Fig. 2: Thesis Structure

Chapter 2 presents a unified replay-based continuous learning framework for spatio-temporal prediction on streaming data. Specifically, the framework encompasses: i) a data integration module to fuse current and historical data by means of spatio-temporal mixup mechanism to alleviate catastrophic forgetting; ii) a spatio-temporal continuous representation learning module to learn holistic spatio-temporal features utilizing five spatio-temporal augmentation methods; iii) a spatio-temporal prediction module to enable effective prediction; iv) extensive experiments to assess pertinent properties of the proposal.

Chapter 3 presents the LightR+ federated trajectory recovery framework. To be specific, we propose a lightweight local trajectory embedding module to improve efficiency without compromising feature extraction capabilities. An intra-domain knowledge distillation module is designed to alleviate catastrophic forgetting. In addition, we propose a meta-knowledge enhanced local-global training scheme to reduce communications costs. Moreover, we conduct extensive experiments to show the effectiveness of the proposal.

Chapter 4 presents the F-TSL federated trajectory similarity learning framework. In particular, F-TSL encompasses three key modules: i) a local trajectory preprocessing and learning module for lightweight trajectory embeddings; ii) a privacy-preserving clustering module to aggregate similar clients aiming at alleviating data heterogeneities; iii) a server-side aggregation module to obtain a global model by aggregating updates on a local cluster; and iv) extensive experiments to show the superiority of the proposal.

Chapter 1

Less is More: Efficient Time Series Dataset Condensation via Two-fold Modal Matching

This chapter provides an overall introduction to Paper A [45], which proposes a time series dataset condensation framework, TimeDC, that employs two-fold modal matching. TimeDC can be used in different downstream tasks, e.g., time series forecasting. Extensive experiments on real data offer insight into the effectiveness and efficiency of the proposed TimeDC. This chapter reuses content from the paper [45] when that is considered most effective.

1 Problem Motivation and Statement

With the proliferation of edge computing and mobile sensing, massive volumes of time series data, comprising millions of observations, are being collected and stored into time series database systems (TSMSs) [31, 67], enabling various real-world applications [54, 65, 73]. However, storing and preprocessing datasets is costly, and model training often calls for the use of specialized equipment and infrastructure [35, 81, 86], limiting the application on edge devices [67, 77]. We consider the novel problem of time series dataset condensation, where the goal is to synthesize a small but informative condensed time series dataset $\mathcal{S}$ derived from an original large dataset $\mathcal{T}$. A model, composed of stacked time series operators (*TSOperators*) that perform feature extraction, is trained on the condensed time series (CTS) dataset with the objective of achieving performance comparable to that of a model trained on the original large time series (OLTS) dataset for downstream tasks (see Figure 1.1).

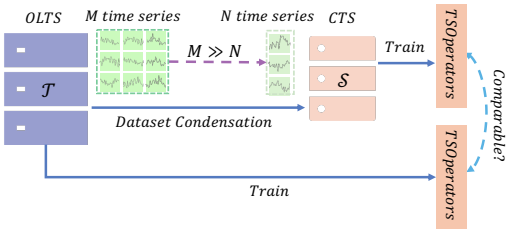


Fig. 1.1: Time Series Condensation

However, developing such methods is non-trivial, due to the following challenges. First, It is challenging to guarantee the effectiveness and generalization ability of the condensed time series dataset [1, 9]. Typical dataset condensation methods require a bi-level (even triple-level) optimization to jointly learn a minimum of two objectives: model parameters and a condensed time series dataset. Such complex non-convex optimization cannot guarantee optimal solutions, thus significantly limiting its effectiveness as the representative of the large original time series dataset. Second, existing popular dataset condensation methods mainly focus on computer vision without specific modules to capture the unique time series characteristics [7, 73]. Third, existing dataset condensation methods [62, 86] often suffer from poor scalability, as bi-level optimization is generally time-consuming [92].

This study addresses the above challenges by providing an efficient time series dataset condensation (TimeDC) framework, which features a two-fold matching mechanism: time series frequency matching and training trajectory matching. TimeDC synthesizes a small time series dataset that summarizes a large dataset such that models trained on the small dataset achieve comparable performance to models trained on the large dataset. In particular, we propose a curriculum training trajectory matching (CT^2M) module to achieve effective and generalized TS dataset condensation. We design a decomposition-driven frequency matching (DDFM) module to facilitate consistent temporal pattern preservation between the condensed time series dataset and the original one. It maps the decomposed frequencies when optimizing the condensed dataset. In addition, to support the capture of complex temporal dependencies (*Challenge II*), we propose a time series feature extraction (TSFE) module. Finally, To address the issue of high computation costs and poor scalability (*Challenge III*), we propose an expert buffer to save expert trajectories pre-computed on original time series data to avoid memory overflow and reduce the overall training time. The main contributions of paper A are summarized as follows.

- To the best of our knowledge, this is the first study to learn dataset condensation over time series. We propose a framework called TimeDC that aims to condense large time series datasets into small synthetic time series datasets while retaining the expressiveness of the large datasets.

- We design a TSFE module to capture temporal dependencies of time series effectively. We further propose a DDFM module to reduce the discrepancy of frequencies between original and condensed time series.
- A novel curriculum trajectory query and matching module is proposed to penalize condensed time series data based on how far the synthetically trained model deviates from expert trajectories.
- We report on experiments using real data, offering evidence of the effectiveness and efficiency of the proposals.

2 Preliminaries

2.1 Basic Concepts

Definition 1 (Time Series) *A time series* $T = \langle t_1, t_2, \cdots, t_n \rangle$ *is a time ordered sequence of n observations, where each observation* $t_i \in \mathbb{R}^C$ *is a C-dimensional vector. If* $C = 1$, T *is univariate; otherwise,* T *is multivariate.*

Definition 2 (Time Series Dataset) *A time series dataset* $\mathcal{T}$ *is a set of time series* $\mathcal{T} = \{T_1, T_2, \cdots, T_M\}$, *where* M *is the cardinality.*

Definition 3 (Condensed Time Series Dataset) *Given a time series dataset* $\mathcal{T}$, *a time series dataset* $\mathcal{S} = \left\{\widetilde{T}_1, \widetilde{T}_2, \cdots, \widetilde{T}_N\right\}$ *is condensed if* N *is much smaller than* M, *where* $\widetilde{T}$ *is the condensed time series,* M *is the cardinality, and* $\mathcal{S}$ *is derived from* $\mathcal{T}$.

2.2 Problem Definition

Given a large time series dataset $\mathcal{T}$, our goal is to synthesize a small time series dataset $\mathcal{S} = \left\{\widetilde{T}_1, \widetilde{T}_2, \cdots, \widetilde{T}_N\right\}$ that preserves most of the information in $\mathcal{T}$ and $N \ll M$. Once the condensed time series set $\mathcal{S}$ is learned, we can train f on it by replacing $\mathcal{T}$ by $\mathcal{S}$ in Equation 1.1.

$$\theta^{\mathcal{T}} = \arg\min_{\theta} \mathcal{L}^{\mathcal{T}}(\theta), \tag{1.1}$$

As the condensed set $\mathcal{S}$ is significantly smaller than $\mathcal{T}$, we expect the optimization to be considerably faster. In addition, we expect the generalization performance of $f_{\theta^{\mathcal{S}}}$ to be close to $f_{\theta^{\mathcal{T}}}$, i.e., $\mathbb{E}_{T_i \sim P_{\mathcal{T}}}[\ell(f_{\theta^{\mathcal{T}}}, T_i)] \simeq \mathbb{E}_{T_i \sim P_{\mathcal{S}}}[\ell(f_{\theta^{\mathcal{S}}}, T_i)]$, on the real data distribution $P_{\mathcal{T}}$. Thus, given a learnable function f parameterized by θ, our problem, *dataset condensation over time series*, can be formulated as a bi-level optimization problem [86] as follows.

$$\overbrace{\min_{\mathcal{S}} \mathcal{L}(f_{\theta^{\mathcal{S}}}, \mathcal{T})}^{outer-level}, \; s.t. \; \theta^{\mathcal{S}} = \overbrace{\arg\min_{\theta} \mathcal{L}(f_{\theta}, \mathcal{S})}^{inner-level}, \tag{1.2}$$

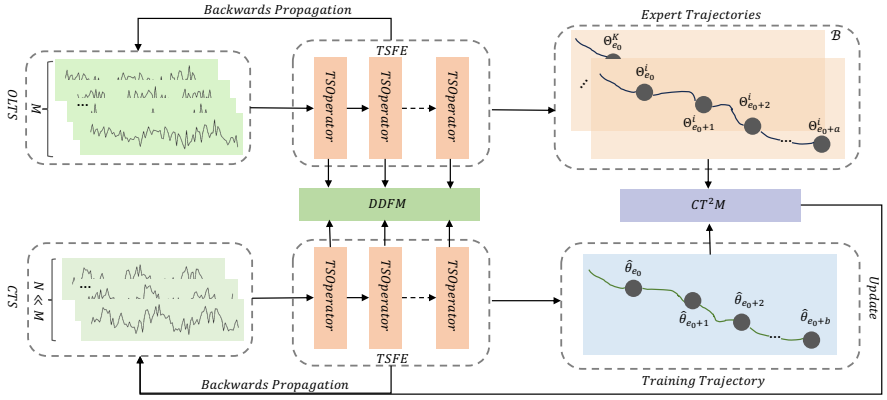


Fig. 1.2: TimeDC Framework Overview

where the outer loop optimizes the condensed time series set $\mathcal{S}$, while the inner loop aims to learn an optimal function f_{θ^S} on $\mathcal{S}$.

3 Methodology

As illustrated in Figure 1.2, given a model composed of stacked *TSOperators*, we first pre-train a set of training trajectories on the original large time series (OLTS) dataset and store these pre-trained trajectories as expert trajectories in an expert buffer. As the original dataset is used to guide the network training, we denote the parameters trained on the original dataset as expert trajectory. Then, we train the same model on the condensed time series (CTS) dataset and perform two-fold modal matching: trajectory matching and frequency matching. Specifically, we optimize the condensed dataset with respect to the distance between the synthetically training trajectories and the trajectories trained on the original dataset. Further, we align the frequencies (i.e., trend and seasonality) between the original dataset and the condensed dataset to preserve temporal correlations when synthesizing the condensed dataset.

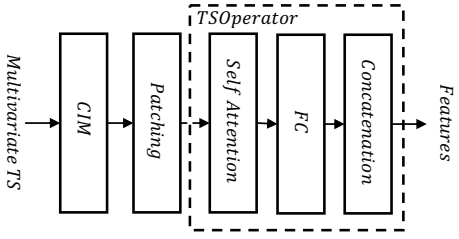


Fig. 1.3: Feature Extraction Module

3.1 Time Series Feature Extraction

We first consider the time series feature extraction (TSFE) module. As shown in Figure 1.3, TSFE is composed of a channel-independent mechanism, a patching mechanism, and stacked *TSOperators*. TSFE aims to extract effective high-dimensional features from the input time series. A channel-independent mechanism $CIM(\cdot)$ is adopted to decouple the time series $T_{input} \in \mathbb{R}^{\mathcal{B}\times n\times C}(C \geq 1)$ into C univariate time series $\{T^c_{input}\}^C_{c=1} \in \mathbb{R}^{\mathcal{B}\times n\times 1}$ for subsequent feature extraction.

Next, the module splits each decoupled univariate (i.e., channel-independent) time series into P patches to learn local semantic information employing the patching mechanism. Given a patch length L and a stride (i.e., the non-overlapping region between two consecutive patches) S, a sequence of patches $T^c_p \in \mathbb{R}^{\mathcal{B}\times L\times \mathcal{P}}$ is generated, where $\mathcal{P} = \lfloor \frac{n-L}{S} + 2 \rfloor$ is the number of patches. The last value t_n repeated $S-1$ times is padded to the end of the input time series before patching.

Then, the module stacks several *TSOperators*, composed of self-attention and fully connected layers, for channel-independent feature extraction. The *j-th TSOperator* can be formulated as follows.

$$h^j = TSOperator(h^{j-1}) = Norm(FC(MultiHead(h^{j-1}))), \tag{1.3}$$

3.2 Decomposition-Driven Frequency Matching

Time series data usually exhibit distinct frequencies, such as trend and seasonality, separating from other modalities (e.g., images). We argue that it would give more guidance on time series condensation if the condensed and original time series share similar frequencies. Thus, to preserve temporal patterns in the condensed time series dataset, we propose a novel decomposition-driven frequency matching (DDFM) mechanism that incorporates a decomposition block into the stacked *TSOperators*.

As shown in Figure 1.2, for each *TSOperator*, DDFM decomposes the intermediate hidden features, learned from the original dataset $\mathcal{T}$ and the condensed dataset $\mathcal{S}$, into their frequencies (i.e., trend and seasonality) progressively. It then aligns the frequencies to reduce the discrepancy of the temporal patterns between the original dataset and the condensed dataset based on the moving average. Concretely, for the hidden features h^j of the *j-th TSOperator*, we adopt the moving average by means of average pooling $AvgPool(\cdot)$ to smooth periodic fluctuations and highlight the trend and seasonality. In particular, the process of series decomposition is as follows.

$$h^j_{TRE} = AvgPool(Padding(h^j)),\ h^j_{SEA} = h^j - h^j_{TRE}, \tag{1.4}$$

where h^j_{TRE} and h^j_{SEA} are the decomposed trend and seasonality, respectively. We employ the average pooling $AvgPool(\cdot)$ with the padding operation

Padding$(\cdot)$ to keep the time series length unchanged. We denote the decomposition process as follows.

$$h^j_{TRE}, h^j_{SEA} = SeriesDecomposition(h^j) \tag{1.5}$$

Given the extracted features $h^j_{\mathcal{T}}$ and $h^j_{\mathcal{S}}$ of the original dataset $\mathcal{T}$ and the condensed dataset $\mathcal{S}$, we decompose the corresponding trend and seasonality according to Equation 1.5, We use cosine similarity $cos(\cdot,\cdot)$ to measure similarities between the trends $h^j_{TRE_{\mathcal{T}}}$, $h^j_{TRE_{\mathcal{S}}}$ and seasonalities $h^j_{SEA_{\mathcal{T}}}$, $h^j_{SEA_{\mathcal{S}}}$, formulated as follows.

$$\begin{aligned} cos(h^j_{TRE_{\mathcal{T}}}, h^j_{TRE_{\mathcal{S}}}) &= \frac{h^j_{TRE_{\mathcal{T}}}}{||h^j_{TRE_{\mathcal{T}}}||_2} \cdot \frac{h^j_{TRE_{\mathcal{S}}}}{||h^j_{TRE_{\mathcal{S}}}||_2} \\ cos(h^j_{SEA_{\mathcal{T}}}, h^j_{SEA_{\mathcal{S}}}) &= \frac{h^j_{SEA_{\mathcal{T}}}}{||h^j_{SEA_{\mathcal{T}}}||_2} \cdot \frac{h^j_{SEA_{\mathcal{S}}}}{||h^j_{SEA_{\mathcal{S}}}||_2}, \end{aligned} \tag{1.6}$$

where $||\cdot||_2$ is l_2-*norm*.

We aim to maximize the cosine similarity to reduce the discrepancy of frequencies between the condensed data and the original data. The objective function of DDFM is formulated as follows.

$$L_{Fre} = -\frac{1}{N_{op}} \sum_{j}^{N_{op}} (cos(h^j_{TRE_{\mathcal{T}}}, h^j_{TRE_{\mathcal{S}}}) + cos(h^j_{SEA_{\mathcal{T}}}, h^j_{SEA_{\mathcal{S}}})), \tag{1.7}$$

where N_{op} is the number of *TSOperators*.

3.3 Curriculum Training Trajectory Matching

To enable effective dataset condensation, we propose a curriculum training trajectory matching mechanism. Motivated by an existing study [9], we match the long-term training trajectories of the TTFM module with the offline guidance of expert trajectories, which are pre-trained and stored in the expert buffer.

In particular, we first train K time series models with the same architecture, i.e., stacked *TSOperators*, denoted as $f_{\mathcal{T}}$, on the original large time series dataset. Then, we can obtain K numbers of expert training trajectories that have a holistic knowledge of the original time series dataset in terms of $f_{\mathcal{T}}$'s training process. We save these model parameters $\{\Theta^k_{\mathcal{T}}\}^K_{k=1} = \{\theta^k_e\}^E_{e=1}$ at certain epoch intervals E in the expert buffer $\mathcal{B}$. Finally, we design a curriculum trajectory query mechanism to sample training trajectories of the expert buffer from similar to dissimilar for trajectory matching. Here, the pre-training of $f_{\mathcal{T}}$ is offline and can be separated from the end-to-end time series condensation, reducing the online computation costs.

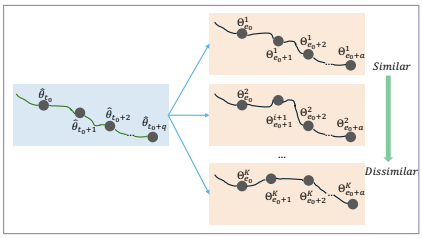


Fig. 1.4: Curriculum Trajectory Query and Matching

Training Trajectory Matching. We use the original dataset to guide the network training, and the parameters trained on this dataset are called an expert trajectory. If a condensed time series dataset is capable of forcing network training dynamics to follow expert trajectories, the idea is that the synthetically trained network will be located close to the model trained on the original time series dataset and will achieve similar test performance for downstream applications.

When sampling a pre-computed parameter trajectory θ_e from the expert buffer $\mathcal{B}$, we aim to minimize the distance between the parameters $\widetilde{\theta}_e$ trained on the condensed dataset and θ_e:

$$\arg\min_{\mathcal{S}} \mathbb{E}_{\theta_e \sim \mathcal{B}}[L_{tmm}(\theta_e|_{e=e_0}^{a}, \widetilde{\theta}_e|_{e=e_0}^{b})], \tag{1.8}$$

where $\mathbb{E}$ is the expectation, $\theta_e|_{e=e_0}^{a}, \widetilde{\theta}_e|_{e=e_0}^{b}$ are the parameters with range $(e_0, e_0 + a)$ and $(e_0, e_0 + b)$, where $e_0 < e_0 + a < E$. More specifically, we update the condensed TS data according to the trajectory matching loss L_{tmm}.

$$L_{tmm} = \frac{||\widetilde{\theta}_{e_0+b} - \theta_{e_0+a}||_2^2}{||\widetilde{\theta}_{e_0} - \theta_{e_0+a}||_2^2}, \tag{1.9}$$

Note that we initialize the parameters of $f_{\theta}s$ with those of $f_{\mathcal{T}}$ at e_0 training step, i.e., $\widetilde{\theta}_{e_0} = \theta_{e_0}$, for more focused training. This way, we align the learning behavior of $f_{\mathcal{T}}$ with a-steps optimization to $f_{\theta}s$ with b-steps optimization, imitating the long-term learning behavior of time series modeling.

For the inner loop, we train $f_{\theta}s$ on the condensed time series data for optimization until the optimal $\widetilde{\theta}^*$. Thus, the objective function of the CT^{2}M is defined as follows.

$$\min_{\mathcal{S}} \mathbb{E}_{\theta_e^* \sim \mathcal{B}}[L_{tmm}(\theta_e|_{e=e_0}^{a}, \widetilde{\theta}_e|_{e=e_0}^{b})] \quad s.t.\ \widetilde{\theta}^{*,\mathcal{S}} = \arg\min_{\widetilde{\theta}} \mathcal{L}(f_{\widetilde{\theta}}, \mathcal{S}), \tag{1.10}$$

where $\mathcal{L}(\cdot)$ is the task-specific loss (e.g., Mean Square Error (MSE)).

Curriculum Training Trajectory Query. As shown in Figure 1.4, we propose a curriculum training trajectory query method to further enhance the

generalization of time series condensation and accelerate the model convergence [69, 89]. Curriculum learning aims to train a model with easy samples first, and then gradually increases the difficulty levels. In our setting, we first query similar training trajectories $\theta_e|_{e=e_0}^{a}$ and $\widetilde{\theta}_e|_{e=e_0}^{b}$, learned on the original and condensed datasets, respectively, to perform trajectory matching. Then, we gradually increase the dissimilarity of the original data training trajectory.

To quantify the similarity between training trajectories of the original dataset $\mathcal{T}$ and the condensed dataset $\mathcal{S}$, we design a minimally interfered retrieval sampling strategy. Specifically, given current parameters $\widetilde{\theta}_{e_0}$ of $f_\theta s$ learned on the condensed dataset, trajectories $\{\Theta_{\mathcal{T}}^{k}\}_{k=1}^{K}$ in buffer $\mathcal{B}$, and a standard task-specific objective function $\min_{\theta s} \mathcal{L}(f_\theta s, \mathcal{S})$, we retrieve trajectories that will be close to $\widetilde{\theta}$ by the update of the foreseen (i.e., future) parameters to select more similar trajectories from $\mathcal{B}$. We update the parameter $\widetilde{\theta}_{e_0}$ for a-steps by gradient matching, as shown in Equation 1.11.

$$\widetilde{\theta}_{e_0+a} = \widetilde{\theta}_{e_0} - \sum_{s=1}^{a}(\alpha \nabla \mathcal{L}(f_\theta s, \mathcal{S})), \tag{1.11}$$

where α is the learning rate. Then, we compute the distance $D(\cdot,\cdot)$ between the foreseen trajectory $\widetilde{\theta}|_{e=e_0}^{a}$ of $\mathcal{S}$ and sub-trajectories $\theta^k|_{e=e_0}^{a}$ of $\{\Theta_{\mathcal{T}}^{k}\}_{k=1}^{K}$ in $\mathcal{B}$ as follows.

$$dis_k = -D(\widetilde{\theta}|_{e=e_0}^{a}, \theta^k|_{e=e_0}^{a}), \tag{1.12}$$

We use the cosine similarity to measure the distance $D(\cdot)$.

$$D(\widetilde{\theta}|_{e=e_0}^{a}, \theta^k|_{e=e_0}^{a}) = \frac{\widetilde{\theta}|_{e=e_0}^{a}}{||\widetilde{\theta}|_{e=e_0}^{a}||_2} \cdot \frac{\theta^k|_{e=e_0}^{a}}{||\theta^k|_{e=e_0}^{a}||_2} \tag{1.13}$$

Finally, we match current model trajectories of $\mathcal{S}$ with the pre-trained trajectories in descending order based on the similarity $\{dis_k\}_{k=1}^{K}$.

3.4 Overall Objective Function

The final loss contains three parts: a task-specific loss $\mathcal{L}$, a frequency matching loss L_{Fre}, and a trajectory matching loss L_{tmm}. We combine them together and the overall loss is as follows.

$$L_{all} = \mathcal{L} + L_{Fre} + L_{tmm}. \tag{1.14}$$

The task-specific loss $\mathcal{L}$ is specific to the particular downstream tasks to achieve a better condensed dataset tailored for the intended use.

4 Experimental Evaluation

4.1 Experimental Setup

Datasets

The experiments are carried out on six widely-used time series datasets, covering four application domains: weather, traffic, economics, and energy.

- **Weather.** The Weather dataset contains 21 indicators of weather (e.g., air temperature and humidity), which are collected in Germany. The data is recorded every 10 minutes.
- **Traffic.** The Traffic dataset contains hourly road occupancy rates obtained from sensors located at San Francisco freeways from 2015 to 2016.
- **Electricity.** The Electricity dataset contains the hourly electricity consumption of 321 clients from 2012 to 2014.
- **ETT.** The ETT dataset includes two hourly-level datasets (ETTh1 and ETTh2) and two 15-minute-level datasets (ETTm1 and ETTm2). Each dataset includes 7 oil and load features of electricity transformers between July 2016 and July 2018.

We choose time series forecasting as a representative downstream task, as it is a popular analytics task.

Baselines

We compare TimeDC with the following existing methods that include coreset construction methods (i.e., Random, Herding [71], and K-Center [26]), and dataset condensation methods (i.e., DC [86] and MTT [9]).

Evaluation Metrics

Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) are adopted as the evaluation metrics, which are defined as follows.

$$MAE = \frac{1}{M}\sum_{m=1}^{M}|\hat{\mathcal{Y}}^m - \mathcal{Y}^m|,\ MSE = \frac{1}{M}\sum_{m=1}^{M}||\hat{\mathcal{Y}}^m - \mathcal{Y}^m||^2, \tag{1.15}$$

where M is the testing data size, $\hat{y}^t$ is the prediction and y^t is the ground truth. The smaller the MAE and the RMSE are, the more accurate method is. We also evaluate the efficiency of the models, including the training and dynamic tensor memory cost.

Table 1.1: Overall Performance Comparison on Seven Datasets

Baseline		Random		K-Center		Herding		DC		MTT		TimeDC		Whole Dataset	
Dataset	*PL*	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE
Weather	96	0.731	1.256	0.452	0.687	0.478	0.677	0.361	0.514	0.295	0.244	**0.257**	**0.188**	*0.239*	*0.182*
	192	0.786	1.302	0.487	0.723	0.512	0.688	0.413	0.527	0.344	0.301	**0.285**	**0.247**	*0.261*	*0.195*
	336	0.794	1.311	0.524	0.756	0.554	0.712	0.444	0.567	0.368	0.328	**0.330**	**0.287**	*0.282*	*0.241*
Traffic	96	0.675	1.125	0.503	0.576	0.483	0.554	0.375	0.603	0.279	0.403	**0.254**	**0.375**	*0.247*	*0.337*
	192	0.712	1.144	0.514	0.604	0.517	0.606	0.432	0.633	0.336	0.442	**0.297**	**0.405**	*0.265*	*0.338*
	336	0.729	1.117	0.523	0.611	0.553	0.654	0.449	0.676	0.355	0.471	**0.312**	**0.423**	*0.297*	*0.360*
Electricity	96	0.421	0.669	0.448	0.583	0.501	0.592	0.376	0.513	0.296	0.283	**0.274**	**0.267**	*0.252*	*0.268*
	192	0.450	0.743	0.476	0.601	0.534	0.628	0.419	0.532	0.315	0.337	**0.285**	**0.294**	*0.239*	*0.255*
	336	0.491	0.853	0.506	0.622	0.569	0.477	0.436	0.544	0.339	0.356	**0.304**	**0.322**	*0.271*	*0.285*
ETTh1	96	0.523	0.745	0.554	0.698	0.536	0.656	0.503	0.442	0.456	0.464	**0.413**	**0.401**	*0.354*	*0.386*
	192	0.557	0.786	0.578	0.722	0.589	0.698	0.552	0.508	0.504	0.471	**0.436**	**0.428**	*0.362*	*0.355*
	336	0.588	0.802	0.604	0.745	0.603	0.723	0.556	0.513	0.498	0.464	**0.447**	**0.431**	*0.409*	*0.387*
ETTh2	96	0.487	0.655	0.589	0.711	0.521	0.589	0.463	0.524	0.388	0.342	**0.368**	**0.271**	*0.324*	*0.255*
	192	0.509	0.673	0.605	0.732	0.553	0.621	0.488	0.536	0.416	0.384	**0.389**	**0.302**	*0.332*	*0.257*
	336	0.524	0.689	0.623	0.744	0.564	0.640	0.505	0.540	0.435	0.455	**0.411**	**0.334**	*0.376*	*0.296*
ETTm1	96	0.743	1.124	0.525	0.492	0.607	0.554	0.603	0.665	0.512	0.453	**0.503**	**0.442**	*0.453*	*0.403*
	192	0.764	1.245	0.566	0.510	0.628	0.571	0.597	0.647	0.563	0.501	**0.512**	**0.465**	*0.464*	*0.432*
	336	0.801	1.128	0.571	0.523	0.644	0.582	0.624	0.668	0.552	0.488	**0.500**	**0.483**	*0.477*	*0.455*
ETTm2	96	0.664	0.795	0.486	0.623	0.524	0.558	0.472	0.535	0.376	0.421	**0.354**	**0.391**	*0.347*	*0.381*
	192	0.687	0.804	0.512	0.643	0.549	0.583	0.488	0.567	0.453	0.479	**0.401**	**0.421**	*0.358*	*0.403*
	336	0.702	0.823	0.558	0.661	0.598	0.624	0.493	0.556	0.473	0.523	**0.453**	**0.474**	*0.406*	*0.435*

4.2 Experimental Results

Overall Performance Comparison

We report the MAE and RMSE values of the methods in Table 1.1. The best performance by an existing method (Random, Herding, K-Center, DC, and MTT) is underlined, and the overall best performance is marked in bold. Whole Dataset indicates training on the original dataset and serves as an approximate upper-bound performance; is marked in italics. We use a set of stacked TSFE modules as the basic forecasting model for each baseline. The following observations are made.

- TimeDC achieves the best results on all datasets across all prediction lengths ($PL \in \{96, 192, 336\}$). TimeDC performs better than the best among the baselines by up to 13.49% and 26.59% in terms of MAE and RMSE, respectively. We observe that the performance improvements obtained by TimeDC on the Weather dataset exceed those on the Traffic, Electricity, ETTh1, and ETTh2 in most cases. This is because the Weather has much more training data than the other datasets. TimeDC trained with more training data results in better results. Though the ETTm1 and ETTm2 datasets have substantial training data, TimeDC performs slightly better than baselines on these two datasets. This is because the temporal patterns in ETTm1 and ETTm2 are highly regular and can be captured easily by existing methods.

- The coreset methods (i.e., Random, K-Center, and Herding), perform worse than the condensation methods (i.e., DC, MTT, and TimeDC), where Random has the worst performance in most cases. This is because coreset

methods are based on heuristic metrics, that makes it hard to select representative observations and guarantee the optimal solutions.

- A popular condensation method, MTT performs the best among the existing methods except when $PL = 96$ on ETTh1, due to its powerful condensation capability that involves matching multi-step parameters.

The experiment offers evidence that TimeDC is more effective than existing coreset and dataset condensation methods.

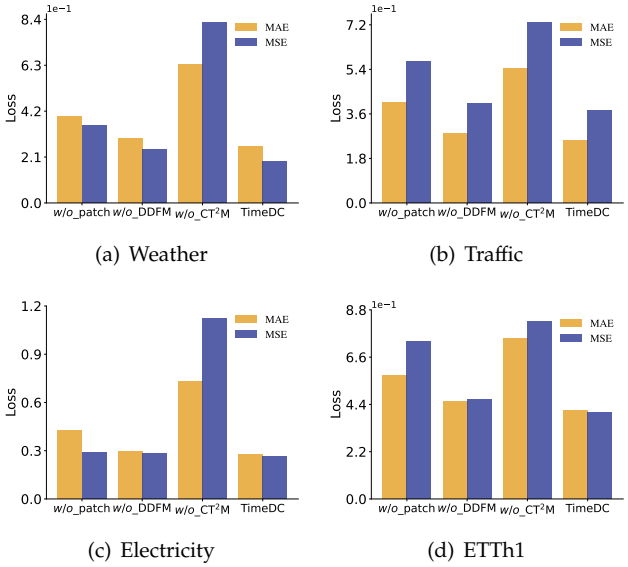


Fig. 1.5: Performance of TimeDC and Its Variants on Four Datasets ($PL = 96$)

Ablation Study

To gain insight into the effects of the different components of TimeDC, including the patching mechanism (patch), decomposition-driven frequency matching (DDFM), and curriculum training trajectory matching (CT^{2}M), we evaluate three variants:

- ***w/o_Patch***. TimeDC without the patching mechanism.
- ***w/o_DDFM***. TimeDC without the DDFM module.
- ***w/o_CT^{2}M***. TimeDC without the CT^{2}M module of TimeDC.

Figure 1.5 shows results on Weather, Traffic, Electricity, and ETTh1. Regardless of the datasets, TimeDC outperforms its counterparts without the patching mechanism, the DDFM module, and the CT^{2}M module. This shows that these

Table 1.2: Cross-Architecture Performance Comparison

Method	Metric	PL	Weather	Traffic	Electricity	ETT			
						ETTh1	ETTh2	ETTm1	ETTm2
TimeDC	MAE	96	**0.257**	**0.254**	**0.274**	**0.413**	**0.368**	**0.503**	**0.354**
		192	**0.285**	**0.297**	**0.285**	**0.436**	**0.389**	**0.512**	**0.401**
	MSE	96	**0.188**	**0.375**	**0.267**	**0.401**	**0.271**	**0.442**	**0.391**
		192	**0.247**	**0.405**	**0.294**	**0.428**	**0.302**	**0.465**	**0.421**
Autoformer	MAE	96	0.312	0.370	0.343	0.453	0.473	0.548	0.342
		192	0.381	0.385	0.355	0.478	0.491	0.550	0.334
	MSE	96	0.255	0.597	0.236	0.465	0.412	0.542	0.265
		192	0.334	0.613	0.264	0.493	0.488	0.532	0.287
Informer	MAE	96	0.423	0.430	0.428	0.773	0.842	0.576	0.552
		192	0.482	0.476	0.446	0.788	0.954	0.597	0.532
	MSE	96	0.354	0.643	0.253	0.992	1.032	0.624	0.402
		192	0.478	0.710	0.271	0.987	1.055	0.653	0.432
Transformer	MAE	96	0.389	0.412	0.398	0.632	0.506	0.563	0.555
		192	0.588	0.431	0.422	0.612	0.513	0.588	0.576
	MSE	96	0.344	0.578	0.267	0.785	0.579	0.615	0.479
		192	0.524	0.567	0.288	0.732	0.542	0.643	0.455

three components are all useful for effective time series dataset condensation. TimeDC obtains MAE and RMSE reductions by up to 12.88% and 22.95%, respectively, compared with w/o_DDFM. Further, on all datasets, w/o_CT^{2}M performs worst among all variants. TimeDC performs better than w/o_CT^{2}M by at least 44.64% and 48.49% in terms of MAE and MSE, respectively, which indicates the effectiveness of the CT^{2}M module.

Cross-Architecture Performance

Next, we consider the cross-architecture performance of TimeDC. It is important to determine whether the condensed time series data generated by TimeDC can be used to train an unseen network. To assess such cross-architecture performance comprehensively, we consider three representative state-of-the-art network architectures for time series forecasting, including *Autoformer* [72], *Informer* [91], and *Transformer* [66]. We first synthesize condensed time series data with TimeDC and then train these networks with the condensed time series data. The parameters of these network architectures are set based on their original papers and any accompanying code. For brevity, we do not include the detailed parameter settings here but refer to the original papers. For TimeDC, we use a set of stacked TSFE modules as the forecasting network.

The prediction results are given in Table 1.2. Overall, TimeDC has the best performance in most cases, indicating its stable and superior performance, especially on Traffic and Electricity. It also demonstrates the effectiveness of the TSFE module. We observe that Autoformer performs worse than TimeDC but better than Informer and Transformer, while the performances of Informer and Transformer are comparable, especially on ETTm1 and ETTm2. These

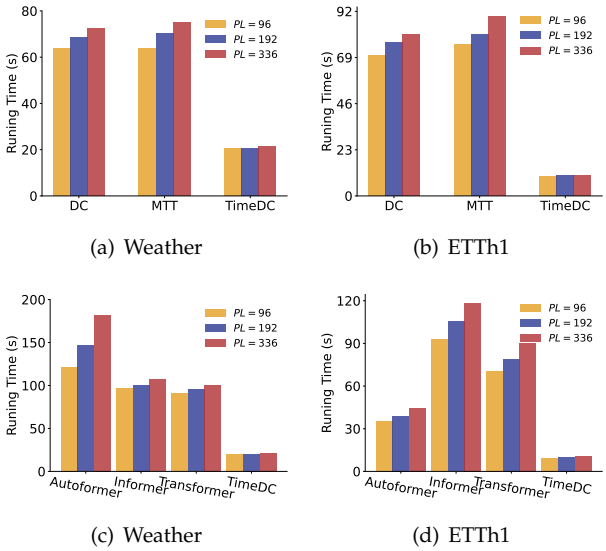
Table 1.3: Dynamic Tensor Memory Cost on Four Datasets

Dataset	DC	MTT	TimeDC
Weather	10.0 GB	8.9 GB	3.3 GB
Traffic	17.8 GB	13.7 GB	10.9 GB
Electricity	8.5 GB	932.5 MB	516.0 MB
ETTh1	1.9 GB	845.5 MB	280.9 MB

observations are in line with their performances when trained on the original dataset [72], indicating that TimeDC learns a generalized condensed time series dataset that works across different network architectures. This is because of the powerful feature extraction capabilities of the proposed stacked *TSOperators*, as well as the trajectory matching that imitates the long-term training dynamics of models trained on the original dataset.

Running Efficiency

As resource efficiency is important in dataset condensation to enable scalability, especially on resource-constrained edge computing devices, we study training time (of an epoch) for the condensation methods. Figures 1.6(a) and 1.6(b) report the training time on Weather and ETTh1. We see that the training time of TimeDC is much lower than those of DC and MTT, which is largely because of the expert buffer in the CT^2M module that stores the pre-computed trajectories. This indicates the feasibility of TimeDC for model deployment in large time series dataset reduction scenarios. We also compare the training time of TimeDC and coreset methods, which are included in the code repository, showing TimeDC has comparable training time with coreset methods. But coreset methods, especially Herding, need more time to construct coreset while having worse performance. Moreover, we compare the memory used by the dynamic (online) tensor across DC, MTT, and TimeDC in Table 1.3. TimeDC is able to reduce markedly the online memory and computation costs thanks to the training trajectories precomputed offline.



(a) Weather (b) ETTh1

(c) Weather (d) ETTh1

Fig. 1.6: Running Efficiency Comparison

Chapter 2

A unified replay-based continuous learning framework for spatio-temporal prediction on streaming data

This chapter gives an overall introduction of Paper B [48]. The chapter reuses content from the paper when that was considered most effective.

1 Problem Motivation and Statement

The continued digitization of societal processes and the accompanying deployment of sensing technologies generate increasingly massive amounts of spatio-temporal data. In this study, we focus on the novel problem of prediction on streaming spatio-temporal data. However, existing spatio-temporal prediction models [23, 34, 59] are trained statically and fail to handle streaming data with the underlying assumption that the old and new datasets follow the same distribution. Direct application of static models to streaming data may cause remarkable performance degradation [55] due to concept drift. As a result, we need a new kind of continuous learning (CL) model that can adapt continually to the data it sees and can keep on learning spatio-temporal prediction tasks over time.

However, it is non-trivial to develop this kind of model, due to the follow-

ing challenges. First, it is challenging to alleviate catastrophic forgetting in CL for spatio-temporal prediction. Catastrophic forgetting is the tendency to abruptly forget previously learned knowledge, which occurs when static models are simply retrained using newly arrived data [55]. Second, it remains a key problem to discover the commonalities of diverse spatio-temporal data and prediction applications. Third, it is challenging for existing models to learn holistic features for spatio-temporal prediction on streaming data. Specifically, holistic features preserve semantic similarities across multiple time periods. In continuous spatio-temporal prediction, preserving previously learned semantic features may facilitate future prediction.

This study addresses the above challenges by providing a Unified Replay-based Continuous Learning (URCL) framework for spatio-temporal prediction on streaming data. URCL encompasses three main modules: data integration, spatio-temporal continuous representation learning (STCRL), and spatio-temporal prediction. To alleviate catastrophic forgetting (Challenge I), we propose a spatio-temporal mixup (STMixup) mechanism to fuse current sampled spatio-temporal observations with selected samples from a replay buffer that stores a subset of previously learned observations. We also propose a ranking-based maximally interfered retrieval sampling strategy to select representative samples from the buffer. To support diverse spatio-temporal data and prediction applications (Challenge II), we discover the commonalities of existing methods that are typically based on an autoencoder architecture. We propose a novel spatio-temporal prediction network including a spatio-temporal encoder (STEncoder) and a spatio-temporal decoder (STDecoder) to capture complex spatio-temporal correlations to enable accuracy. To address the issue of holistic-feature preservation (Challenge III), we propose a spatio-temporal simple siamese (STSimSiam) network to avoid holistic feature loss in the STCRL module.

The major contributions are summarized as follows.

- To the best of our knowledge, this is the first study to systematically investigate continuous learning for spatio-temporal prediction on streaming data. We propose a unified replay-based continuous learning framework entitled URCL, where a replay buffer and a spatio-temporal mixup mechanism are designed to alleviate catastrophic forgetting.
- To contend with the diversity of spatio-temporal data and prediction applications, we design a spatio-temporal autoencoder including an STEncoder and an STDecoder for effective spatio-temporal feature learning and prediction.
- A novel STSimSiam network is designed to make learned features in latent spaces more holistic, by applying mutual information maximization to preserve features for continuous spatio-temporal prediction. We

also provide five data augmentation methods by considering unique spatiotemporal properties.

- We report on experiments using real datasets, offering evidence of the effectiveness of the proposed URCL.

2 Preliminaries

2.1 Basic Concepts

Definition 4 (Sensor Network) *A sensor network is denoted by a graph* $G = (V, E)$*, where* V *is a sensor node set, and* E *is an edge set. Each node* $v_i \in V$ *represents a sensor, and each edge* $e_{i,j} \in E$ *indicates connectivity between sensors* v_i *and* v_j*.*

Definition 5 (Spatio-temporal Observation) *Given a sensor network* G*, a spatio-temporal observation collected by all sensors at time slot* t *(e.g., 9:00 a.m.–9:15 a.m.) with a sampling interval* Δt *(e.g., 15 minutes) is denoted by* $X_t \in \mathcal{R}^{|V| \times C}$*, where* $|V|$ *denotes the number of sensors and* C *is the dimensionality of node features (e.g., traffic volume and speed).*

In the rest of the paper, we will use *observation* place of *spatio-temporal observation* when meaning is clear from the context.

Definition 6 (Streaming Spatio-temporal Data Sequence) *Given a sensor network* G *and a time period* $\mathbb{T}_i$ *containing* n *consecutive time slots, i.e.,* $\mathbb{T}_i = \left\langle t_i^1, t_i^2, \cdots, t_i^n \right\rangle$*, a streaming spatio-temporal data sequence is a sequence of matrices, each representing an observation at a specific time slot* t_i^j *with one sampling interval, where* $t_i^{j+1} - t_i^j = \Delta t (1 \leq j \leq n-1)$*. In particular, a streaming spatio-temporal data sequence* D_i *is a sequence of observations* $D_i = \left\langle X_{t_i^1}, X_{t_i^2}, \cdots, X_{t_i^n} \right\rangle$*, where* n *is also the sequence length.*

2.2 Problem Definition

Consider a sensor network G that emits a sequence $\mathbb{D} = \langle D_1, D_2, \cdots, D_m \rangle$ ($m \geq 1$) of streaming spatio-temporal data sequences, where D_i denotes a sequence of observations during time period $\mathbb{T}_i$ and $|D_i| = n$ (e.g., $\mathbb{T}_i$ is a day, and n is 96 when having 15min interval). Given a current observation $X_{t_i^j}$ that is contained in D_i ($1 \leq i \leq m$), the SSTP problem aims to learn a function $f_i(\cdot)$ to predict N future observations based on the current observation and its previous $M-1$ ($M < n$) historical observations while maximally preserving the

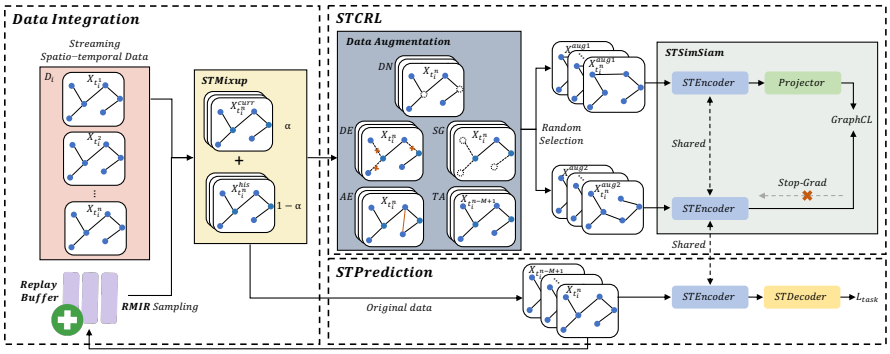


Fig. 2.1: URCL Framework Overview

learned knowledge from previous streaming data sequences $\langle D_1, \cdots, D_{i-1} \rangle$, i.e.,

$$[\overbrace{\cdots, X_{t_i^{j-1}}, X_{t_i^j}}^{M \text{ observations}}; G] \xrightarrow{f_i(\cdot)} [\overbrace{X_{t_i^{j+1}}, X_{t_i^{j+2}, \ldots}}^{N \text{ observations}}], \tag{2.1}$$

where the learned knowledge from $\langle D_1, \cdots, D_{i-1} \rangle$ is preserved maximally.

3 Methodology

We propose a framework, namely Unified Replay-based Continuous Learning (URCL), for spatio-temporal prediction on streaming data. The framework consists of three major modules: data integration, spatio-temporal continuous representation learning (STCRL) and spatio-temporal prediction (STPrediction), as shown in Figure 2.1.

3.1 Data Integration

Considering that we feed data into the framework in a stream, we first sample data from the current dataset D_i and sample historical data from a replay buffer, which stores a subset of previously learned observations, by a ranking-based maximally interfered retrieval (RMIR) sampling strategy. Then we integrate the two kinds of data with the help of the proposed spatio-temporal mixup (STMixup) mechanism, to goal being to better accumulate the spatio-temporal knowledge and alleviate catastrophic forgetting.

To sample more representative samples from replay buffer $\mathcal{B}$, we design a novel ranking-based maximally interfered retrieval (RMIR) sampling method. Given the current data sequence D_i, we first sample M observations

$\mathcal{X}_M = \left\langle X_{t_i^{k-M+1}}, \cdots X_{t_i^k} \right\rangle$ starting at time slot t_i^{k-M+1} from D_i and select samples $\mathcal{X}_\mathcal{B}$ stored in the replay buffer $\mathcal{B}$, where $\mathcal{B}$ is designed to act as an explicit memory to maintain a subset of previously learned observations (i.e., the previously trained observations without STMixup). Then, the selected samples are combined with the current observations through a proposed spatio-temporal mixup (STMixup) mechanism to alleviate catastrophic forgetting by benefiting from historical observations. We can formulate the process of data integration as follows.

$$\begin{aligned} \mathcal{X}_\mathcal{B} &= RMIR(\mathcal{B}, size = |\mathcal{S}|) \\ \mathcal{X}_{mix} &= STMixup(\mathcal{X}_M, \mathcal{X}_\mathcal{B}), \end{aligned} \tag{2.2}$$

where $|\mathcal{S}|$ is the sampling size.

RMIR Sampling Method

We design an RMIR sampling method to select $|\mathcal{S}|$ representative observations $\mathcal{X}_\mathcal{B}$ from the replay buffer. Typically, most methods in existing replay-based continuous learning methods randomly select observations from the replay memory, which leads to unsatisfied results about accuracy, due to the fact that they may ignore representative observations for replay. To select more representative samples, we first retrieve $|\mathcal{N}|$ observations $\mathcal{X}_\mathcal{N}$ which will be the most negatively impacted that suffer from an increase in loss by the update of foreseen parameters, where $|\mathcal{N}| > |\mathcal{S}|$. Specifically, given observation $X_{t_i^\mathcal{B}}$ in $\mathcal{B}$ and a standard objective function $\min_\theta \mathcal{L}_{RMIR}(f_\theta(X_{t_i^\mathcal{B}}), Y_{t_i^\mathcal{B}})$, where $f_\theta(\cdot)$ represents the URCL model, we update the parameters θ from $X_{t_i^\mathcal{B}}$ by gradient descent, as shown in Equation 2.3.

$$\theta^v = \theta - \alpha \bigtriangledown \mathcal{L}_{RMIR}(f_\theta(X_{t_i^\mathcal{B}}), Y_{t_i^\mathcal{B}}), \tag{2.3}$$

where $\mathcal{L}_{RMIR}$ denotes sampling loss, $f_\theta(\cdot)$ represents the model, and $Y_{t_i^\mathcal{B}}$ is the ground truth. Note that Mean Absolute Error (MAE) is adopted as sampling loss. We select the top-$\mathcal{N}$ values.

Considering temporal correlations (e.g., trend and periodicity) of spatio-temporal data, data from a long time ago (periodic data) have a significant impact on the current prediction due to similarities. We then calculate the similarities between observations in $\mathcal{X}_\mathcal{N}$ and $\mathcal{X}_M$ by Pearson coefficient. Finally, we sample top-$\mathcal{S}$ observations of $\mathcal{X}_\mathcal{N}$ that are most similar to $\mathcal{X}_M$. In this way, we can not only select samples that can alleviate catastrophic forgetting, but also select samples to enhance temporal dependency capturing.

STMixup Mechanism

To benefit from historical observations, we introduce STMixup mechanism to fuse current observations $\mathcal{X}_M$ and observations $\mathcal{X}_{\mathcal{B}}$ sampled in $\mathcal{B}$. In particular, STMixup interpolates between $\mathcal{X}_M$ and $\mathcal{X}_{\mathcal{B}}$ to encourage the model to behave linearly across streaming spatio-temporal data sequences to minimize catastrophic forgetting.

Typically, assuming that (x_i, y_i) and (x_j, y_j) are two randomly selected feature-target pairs in the training data, STMixup generates virtual training examples by interpolation based on the principle of Vicinal Risk Minimization [12] to enlarge the support of the training distribution that overcomes concept drift. In STMixup, we use observation-groundtruth pairs to represent feature-target pairs in training. We use $(\tilde{x}, \tilde{y})$ to denote the interpolated feature-target pair in the vicinity of the raw two pairs.

$$\begin{aligned} \tilde{x} &= \lambda \cdot x_i + (1 - \lambda) \cdot x_j \\ \tilde{y} &= \lambda \cdot y_i + (1 - \lambda) \cdot y_j, \end{aligned} \tag{2.4}$$

where $\lambda \sim Beta(\alpha, \alpha)$, and $\alpha \in (0, \infty)$. We design STMixup by interpolating between current observations $\mathcal{X}_M$ and observations $\mathcal{X}_{\mathcal{B}}$ sampled from the replay buffer $\mathcal{B}$ by RMIR sampling. The interpolated observations $\mathcal{X}_{mix}$ after STMixup are formulated as follows:

$$\mathcal{X}_{mix} = \lambda \cdot \mathcal{X}_M + (1 - \lambda) \cdot \mathcal{X}_{\mathcal{B}} \tag{2.5}$$

The interpolated observations $\mathcal{X}_{mix}$ can enhance a model's ability to learn continuously by revisiting past instances in the replay buffer $\mathcal{B}$, that would be most negatively impacted by foreseen parameters. Moreover, STMixup can introduce an approximation of a regularized loss minimization [85] to avoid overfitting.

3.2 Spatio-temporal Continuous Representation Learning

In this module, a spatio-temporal SimSiam (STSimSiam) network, which is a variant of self-supervised Siamese networks [16], is adopted for holistic representation learning by means of mutual information maximization. For the assistance of self-supervised learning, we propose five different data augmentation methods, i.e., DropNodes (DN), DeleteEdges (DE), SubGraph (SG), AddEdge (AE), and TimeShifting (TS), based on the specific characteristics of spatio-temporal data. Then, the augmented data generated by two randomly selected augmentation methods are inserted into two spatio-temporal encoders (STEncoders) that share parameters with each other, followed by a projection MLP (projector).

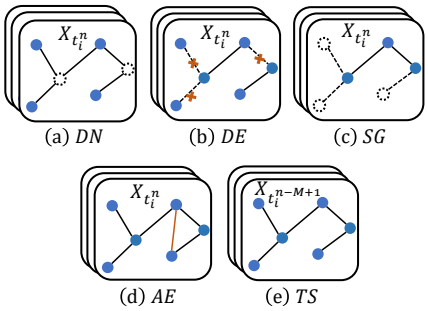


Fig. 2.2: Spatio-temporal Data Augmentation

The interpolated observations $\mathcal{X}_{mix}$ are input into the spatio-temporal continuous representation learning (STCRL) module for holistic feature learning. STCRL is a carefully designed self-supervised learning module, which consists of two parts: spatio-temporal data augmentation and an STSimsiam network. To enable effective spatio-temporal learning, we propose five customized spatio-temporal data augmentation methods by considering unique properties of spatio-temporal data, which transform a sample (i.e., observations in a sensor network) $\mathcal{G} = [\mathcal{X}_{mix}; G]$ to its corresponding perturbation $\mathcal{G}'$. We randomly select two different perturbations $\mathcal{G}'_1$ and $\mathcal{G}'_2$, and then input them into an STSimSiam network, which consists of two STEncoders $f_{\theta_{STE}}$ and a projection MLP head $h(\cdot)$. The aim of STSimSiam is to better capture spatio-temporal dependencies. Finally, STSimSiam maximizes the mutual infomation between the representations of two selected perturbations to ensure holistic feature preservation.

We proceed to elaborate the *spatio-temporal data augmentation* and the *STSimSiam network*.

Spatio-Temporal Data Augmentation

We augment the interpolated observations $\mathcal{X}_{mix}$ using five carefully designed spatio-temporal augmentation methods, which builds semantically similar pairs and improves the quality of the learned representations by defeating perturbations. We propose five spatially oriented data augmentation methods (i.e., DropNodes (DN), DropEdge (DE), SubGraph (SG), and AddEdge (AE)) and a temporally oriented method (i.e., TimeShifting (TS)). We cover each method next.

- **DN.** As shown in Figure 2.2(a), given a sample $\mathcal{G} = [\mathcal{X}_{mix}; G]$, DN randomly discards a certain proportion (e.g., 10%) of the nodes in G to get $\mathcal{G}' = [\mathcal{X}_{mix}; G']$ following a definite distribution (e.g., a uniform distribution), which ensures that the missing nodes have no impact on the

semantics (e.g., distribution) of G.

$$A'_{i,j} = \begin{cases} 0, & \text{if } v_i \text{ is discarded} \\ A_{i,j}, & \text{otherwise,} \end{cases} \tag{2.6}$$

where $A'_{i,j}$ is an entry of the augmented matrix A'. Ideally, DN can promote model robustness by being less affected by missing data caused, e.g., sensor or communication failures.

- **DE.** DE randomly drops a part of edges, as shown in Figure 2.2(b). We first sample a certain ratio of edges $\mathcal{E}$ from G following a specific distribution, and then set a threshold θ_{DE}. If the weights of edges in $\mathcal{E}$ are lower than θ_{DE}, we delete the corresponding edges, formulated as follows.

$$a'_{i,j} = \begin{cases} 0, & \text{if } \mathrm{a_{i,j}} < \theta_{\mathrm{DE}} \\ a_{i,j}, & \text{otherwise,} \end{cases} \tag{2.7}$$

where $a_{i,j}$ denotes the weight of the edge between nodes v_i and v_j, and $a'_{i,j}$ is the updated weight. The aim of the threshold is to retain important connectives of edges.

- **SG.** SG samples a subgraph $G' = (V', E')$ from $G = (V, E)$ by random walk to maximally preserve the semantics of a sensor network (cf. Figure 2.2(c)), where $V' \subseteq V$ and $E' \subseteq E$.

- **AE.** AE randomly selects a certain ratio of distant node pairs (e.g., more than three hops) and add edges between each node pair, which is shown in Figure 2.2(d). Considering a node pair (v_i, v_j), the corresponding weight $w_{i,j}$ is calculated as follows.

$$w_{i,j} = \vec{\mathcal{X}}^i_{mix} \cdot \vec{\mathcal{X}}^j_{mix}, \tag{2.8}$$

where $\vec{\mathcal{X}}^i_{mix}$ is a vector that represents features of node v_i.

- **TS.** As shown in Figure 2.2(e), TS, including time slicing, time warping, and time flipping, transforms current observations $\mathcal{X}_{mix}$ in $\mathcal{G}$ in the time domain. Note that we randomly select one of TS for model training.

  1. ***Time Slicing.*** Time slicing sub-samples the current observations $\mathcal{X}_{mix}$ in the time domain by randomly extracting continuous slice $\mathcal{X}^{slice}_{mix}$ with length l. Formally,

$$\mathcal{X}^{slice}_{mix} = \left\langle X_{mix,t_i^{s-l+1}}, \cdots, X_{mix,t_i^s} \right\rangle, \tag{2.9}$$

  where $t_i^{k-M+1} \leq t_i^{s-l+1} \leq t_i^s \leq t_i^k$, t_i^k is current time slot, M is the length of observations, and X_{mix,t_i^s} denotes observations after STMixup at time slot t_i^s

2. ***Time Warping.*** Time warping upsamples sliced observations $\mathcal{X}_{mix}^{slice}$ by linear interpolation to generate warped observations $\mathcal{X}_{mix}^{warp}$, the length of it is equal to that of $\mathcal{X}_{mix}$.

$$\mathcal{X}_{mix}^{warp} = \left\langle X'_{mix,t_i^{k-M+1}}, \cdots, X'_{mix,t_i^k} \right\rangle, \tag{2.10}$$

where X'_{mix,t_i^k} represents the generated observations at time slot t_i^k.

3. ***Time Flipping.*** Time flipping $TF(\cdot)$ flips the sign of warped observations $\mathcal{X}_{mix}^{warp}$ to generate a new sequence $\mathcal{X}_{mix}^{flip}$ in time domain as follows.

$$\mathcal{X}_{mix}^{flip} = \left\langle X'_{mix,t_i^k}, \cdots, X'_{mix,t_i^{k-M+1}} \right\rangle, \tag{2.11}$$

We randomly apply two different data augmentation methods to the integrated observations $\mathcal{X}_{mix}$ (generated by STMixup) to obtain two augmented observations $\mathcal{X}_{mix}^{aug1}$ and $\mathcal{X}_{mix}^{aug2}$.

STSimSiam Network

Inspired by exciting feature learning ability of self-supervised learning, we design a novel STSimSiam network under the guidance of self-supervised learning to capture holistic spatio-temporal representations, inputting two randomly augmented observations $\mathcal{X}_{mix}^{aug1}$ and $\mathcal{X}_{mix}^{aug2}$. More specifically, STSimSiam consists of two STEncoders to learn spatio-temporal representations of $\mathcal{X}_{mix}^{aug1}$ and $\mathcal{X}_{mix}^{aug2}$, respectively, and a projection head to project latent embeddings of $\mathcal{X}_{mix}^{aug1}$ into the latent space of $\mathcal{X}_{mix}^{aug2}$. Finally, considering that mutual information maximization is proven to learn holistic features and thus improves continuous learning [28], we maximize the mutual information between the representations learned from $\mathcal{X}_{mix}^{aug1}$ and $\mathcal{X}_{mix}^{aug2}$ by GraphCL loss.

As shown in the upper-right corner of Figure 2.1, we input two randomly augmented observations $\mathcal{X}_{mix}^{aug1}$ and $\mathcal{X}_{mix}^{aug2}$ into the STSimSiam network. We first encode the two augmented observations as two fixed vectors z_1 and z_2 by STEncoders f_{θ_E} which are made of a specific spatio-temporal network, i.e., GraphWaveNet in this work, to capture the complex spatio-temporal dependencies. Next, we feed z_1 into a projection MLP head to map it into the latent space of z_2, which contains several MLP layers denoted as $h(\cdot)$. The process can be formulated as follows.

$$z_1 = f_{\theta_E}(\mathcal{X}_{mix}^{aug1}), \quad p_1 = h(z_1), \quad z_2 = f_{\theta_E}(\mathcal{X}_{mix}^{aug2}) \tag{2.12}$$

We use stopgrad operation $SG(\cdot)$ [40] to prevent the trivial solution obtained by STSimSiam. For example, $SG(z_2)$ denotes that the STEncoder on $\mathcal{X}_{mix}^{aug2}$ receives no gradient from z_2.

To enhance holistic feature preservation, we employ mutual information maximization to maximize the similarities between representations of $\mathcal{X}_{mix}^{aug1}$ and $\mathcal{X}_{mix}^{aug2}$. First, for the current two augmented observations, we use cosine similarity to measure similarities $C(\cdot)$ between their output vectors p_1 and z_2, formulated as follows.

$$\begin{aligned} C(p_1, z_2) &= \frac{p_1}{||p_1||_2} \cdot \frac{SG(z_2)}{||SG(z_2)||_2} \\ &= \frac{h(f_\theta \mathcal{X}_{mix}^{aug1})}{||h(f_\theta(\mathcal{X}_{mix}^{aug1})||_2} \cdot \frac{SG(f_\theta(\mathcal{X}_{mix}^{aug2}))}{||SG(f_\theta(\mathcal{X}_{mix}^{aug2}))||_2}, \end{aligned} \tag{2.13}$$

where $||\cdot||_2$ is l_2-$norm$.

We let $(\mathcal{X}_{mix}^{aug1}, \mathcal{X}_{mix}^{aug2})$ denote an augmented observation pair. For a minibatch of $\mathcal{S}$ augmented observation pairs, we adopt a GraphCL [79] loss to maximize their mutual information. The GraphCL loss L_{ssl}^s for the s-th augmented observation pair is defined as follows.

$$L_{ssl}^s = -log \frac{exp(C(p_{s,1}, z_{s,2})/\tau)}{\sum_{s'=1, s' \neq s}^{\mathcal{S}} exp(C(p_{s,1}, z_{s',2})/\tau)}, \tag{2.14}$$

where $p_{s,1}$ and $z_{s,2}$ denotes the output vectors of the s-th augmented observation pair $(\mathcal{X}_{mix,s}^{aug1}, \mathcal{X}_{mix,s}^{aug2})$, $C(\cdot)$ is cosine similarity, and τ is the temperature parameter. To extract more effective features [16], we define a symmetric similarity function and obtain final L_{ssl}^s.

$$L_{ssl}^s = -log \frac{exp((\frac{1}{2}C(p_{s,1}, z_{s,2}) + \frac{1}{2}C(p_{s,2}, z_{s,1}))/\tau)}{\sum_{s'=1, s' \neq s}^{\mathcal{S}} exp((\frac{1}{2}C((p_{s,1}, z_{s',2}) + \frac{1}{2}C(p_{s,2}, z_{s',1}))/\tau)}, \tag{2.15}$$

where $p_{s,2} = h(f_{\theta_E}(\mathcal{X}_{mix,s}^{aug2}))$ and $z_{s,1} = f_{\theta_E}(\mathcal{X}_{mix,s}^{aug1})$.

The final GraphCL loss is computed across all augmented pairs in the minibatch as follows.

$$L_{ssl} = \frac{1}{\mathcal{S}} \sum_{s=1}^{\mathcal{S}} L_{ssl}^s \tag{2.16}$$

where $\mathcal{S}$ is the batch size.

3.3 Spatio-Temporal Prediction

We use an STEncoder that shares the same parameters with that in STCRL to learn latent hidden features from the original data, where the learned data are

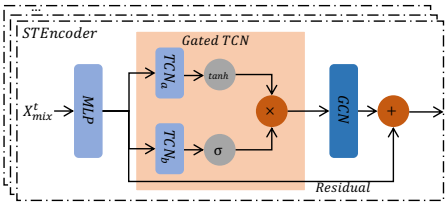


Fig. 2.3: Illustation of STEncoder

then stored in the replay buffer. Finally, the learned features are input into a spatio-temporal decoder (STDecoder) for prediction.

We design a spatio-temporal prediction network including a spatio-temporal encoder (STEncoder) and a spatio-temporal decoder (STDecoder). In particular, we input the interpolated observations $\mathcal{X}_{mix}$ into STEncoder $f_{\theta_E}(\cdot)$ to learn high-dimensional representations through capturing complex spatio-temporal dependencies. The STEncoder shares parameters with the STEncoder in STSimSiam. The learned representations are input into STDecoder f_{θ_D} for prediction. Meanwhile, the recently learned data are stored in the replay buffer. The process can be formulated as follows.

$$h_\theta = f_{\theta_E}(\mathcal{X}_{mix}), \quad \hat{\mathcal{Y}} = f_{\theta_D}(h_\theta), \tag{2.17}$$

where $\hat{\mathcal{Y}}$ is the prediction.

One of the advantages of our framework is its generality. It can easily serve as a plug-in for most existing spatio-temporal prediction models that follow the autoencoder architecture. GraphWaveNet [75] is one of the state-of-the-art spatio-temporal prediction models to capture precise spatial dependencies and long-term temporal dependencies, but which is not developed under the autoencoder architecture. Inspired by its excellent performance on deep spatio-temporal graph modeling, we take it as an example of spatio-temporal prediction models in our work to show how to reorganize its architecture so as to conform to the autoencoder architecture (i.e., STEncoder and STDecoder).

STEncoder

The architecture of STEncoder is shown in Figure 2.3. Taking $\mathcal{X}_{mix}$ as input, an MLP layer maps it into a high-dimensional latent space, and the learned features are then input into a Gated Temporal Convolution Network (TCN) layer to learn temporal correlations among observations of the input. Next, a Graph Convolutional Network (GCN) layer is used to capture spatial correlations among the observations, where a residual operation is used to ensure accuracy. For brevity, we refer to existing studies [48, 75] for more information regarding Gated TCN and GCN.

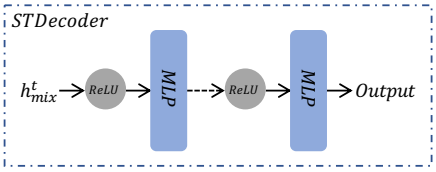


Fig. 2.4: Illustation of STDecoder

STDecoder

The learned features are then input into the STDecoder to decode the data representations for prediction. As shown in Figure 2.4, the STDecoder contains several stacked feed-forward layers (i.e., MLPs) followed by activation functions (i.e., ReLU) to learn a projection function for future prediction. It can be formulated as follows.

$$\hat{\mathcal{Y}} = W_{\theta_D}(\alpha(h_\theta) + b_{\theta_D})), \tag{2.18}$$

where $\hat{\mathcal{Y}}$ denotes the prediction, W_{θ_D} is a learnable parameter, $\alpha(\cdot)$ is the ReLU activiation function, and b_{θ_D} is the bias.

3.4 Overall Objective Function

The overall objective of URCL is to minimize the prediction error by Mean Absolute Error (MAE) for each data stream D_i. The objective function is given in the following:

$$L_{task} = \frac{1}{\mathcal{L}} \sum_{l=1}^{\mathcal{L}} |\hat{\mathcal{Y}}^l - \mathcal{Y}^l|, \tag{2.19}$$

where $\mathcal{L}$ is the training sample size, $\hat{\mathcal{Y}}^l$ is the prediction, and $\mathcal{Y}^l$ is the ground truth.

The final loss contains two parts: a prediction loss of the prediction task L_{task} and a GraphCL loss L_{ssl} for holistic feature learning. We combine them together and the overall loss is as follows.

$$L_{all} = L_{task} + L_{ssl} \tag{2.20}$$

Table 2.1: *Performance of Training on Streaming Data on Two Datasets*

Method	PEMS-BAY										PEMS08									
	MAE					RMSE					MAE					RMSE				
	$\mathcal{B}_{set}$	$\mathcal{I}^1_{set}$	$\mathcal{I}^2_{set}$	$\mathcal{I}^3_{set}$	$\mathcal{I}^4_{set}$	$\mathcal{B}_{set}$	$\mathcal{I}^1_{set}$	$\mathcal{I}^2_{set}$	$\mathcal{I}^3_{set}$	$\mathcal{I}^4_{set}$	$\mathcal{B}_{set}$	$\mathcal{I}^1_{set}$	$\mathcal{I}^2_{set}$	$\mathcal{I}^3_{set}$	$\mathcal{I}^4_{set}$	$\mathcal{B}_{set}$	$\mathcal{I}^1_{set}$	$\mathcal{I}^2_{set}$	$\mathcal{I}^3_{set}$	$\mathcal{I}^4_{set}$
OneFitAll	1.21	2.76	2.72	3.75	3.63	2.17	5.82	4.88	7.28	6.86	18.15	38.75	39.32	44.22	27.21	26.28	44.29	57.34	63.01	37.19
FinetuneST	1.21	3.57	3.57	3.48	3.52	2.16	7.68	7.59	7.66	7.58	18.17	32.60	33.75	34.13	33.15	26.99	37.63	39.79	33.29	35.67
URCL	**1.12**	**1.09**	**1.21**	**1.19**	**1.15**	**1.91**	**1.86**	**2.08**	**2.01**	**1.99**	**18.13**	**18.07**	**17.45**	**17.38**	**17.20**	**26.44**	**26.02**	**25.65**	**24.86**	**24.52**

4 Experimental Evaluation

4.1 Experimental Setup

Datasets

The experiments are carried out on four widely-used graph-based urban spatio-temporal datasets: *METR-LA*, *PEMS-BAY*, *PEMS04* and *PEMS08*, where *METR-LA* and *PEMS-BAY* are related to traffic speed prediction, and *PEMS04* and *PEMS08* are related to traffic flow prediction.

We choose traffic prediction as a representative case of spatio-temporal prediction, as traffic prediction is a popular spatio-temporal prediction tasks.

Evaluation Methods

We compare URCL with six baseline methods: *ARIMA* [58], *DCRNN* [34], *STGCN* [80], *MTGNN* [74], *AGCRN* [4], and *STGODE* [23].

Metrics

Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) are adopted as the evaluation metrics, which are defined as follows.

$$\begin{aligned} MAE &= \frac{1}{M}\sum_{m=1}^{M}|\hat{\boldsymbol{\mathcal{Y}}}^m - \boldsymbol{\mathcal{Y}}^m| \\ RMSE &= \sqrt{\frac{1}{M}\sum_{m=1}^{M}||\hat{\boldsymbol{\mathcal{Y}}}^m - \boldsymbol{\mathcal{Y}}^m||^2}, \end{aligned} \tag{2.21}$$

where M is the testing data size, $\hat{y}^t$ is the prediction and y^t represents the ground truth.

4.2 Experimental Results

Performance of Training on Streaming Data

Our URCL proposes a replay-based strategy for training on streaming data, where a replay buffer is given to store previously learned samples that fuse

with the training data by a spatio-temporal mixup mechanism to preserve historical knowledge effectively. To evaluate the performance of training on streaming data, two representative training strategies are used to replace the replay-based strategy in URCL and are compared with URCL: 1) ***OneFitAll*** that trains a model with the base set and then predicts all the testing data in the stream of spatio-temporal datasets; and 2) ***FinetuneST*** that trains an initial model on the base set and then fine-tunes the model with incremental sets repeatedly. GraphWaveNet is used as the base model of ***OneFitAll*** and ***FinetuneST***. We report the MAE and RMSE results on the PEMS-BAY and the PEMS08 datasets in Table 2.1, where the overall best performance is marked in bold. To save space, we do not report results on the METR-LA and PEMS04 datasets, as these are similar to those obtained for PEMS-BAY and the PEMS08.

We can see that URCL shows the best performance in terms of MAE and RMSE when compared with other methods on both datasets. More specifically, URCL outperforms the best among the baselines by 14.5%–67.3% and 15.5%–72.4% for MAE and RMSE on PEMS-BAY, respectively, and 1.2%–49.1% and 8.9%–35.5% for MAE and RMSE on PEMS08, respectively. OneFitAll and FinetuneST offer acceptable MAE and RMSE results on the base sets on both datasets, but their performance deteriorates on incremental sets. OneFitAll shows that as time goes, the new data can be different from the training data, showing that concept drift happens and a static model does not work, calling for a CL model. While a simple CL based FinetuneST is not enough, as it has forgetting problems. URCL achieves relatively stable performance on both base sets and incremental sets demonstrating its superiority.

Overall Accuracy

We report the MAE and RMSE values of the methods in Table 2.2. The best performance by an existing method (ARIMA, DCRNN, STGCN, MTGNN, AGCRN, and STGODE) is underlined, and the overall best performance is marked in bold. Note that we repeatably train each original baseline on each base and incremental to enable fair comparison with the replay-based strategy. The observations are in the following.

- The proposed URCL achieves the best results among all baselines in most cases, performing better than the best among the baselines by up to 36.0% and 34.1% in terms of MAE and RMSE, respectively. In most cases, URCL outperforms the best among the baselines in terms of MAE and RMSE on the traffic speed datasets (i.e., METR-LA and PEMS-BAY) except for $\mathcal{I}_{set}^{4}$ on METR-LA. Moreover, URCL also outperforms the best among the baselines in terms of MAE and RMSE on the traffic flow datasets (i.e., PEMS04 and PEMS08) except for $\mathcal{I}_{set}^{1}$ on PEMS08. We observe that the performance improvements obtained by URCL on the traffic speed

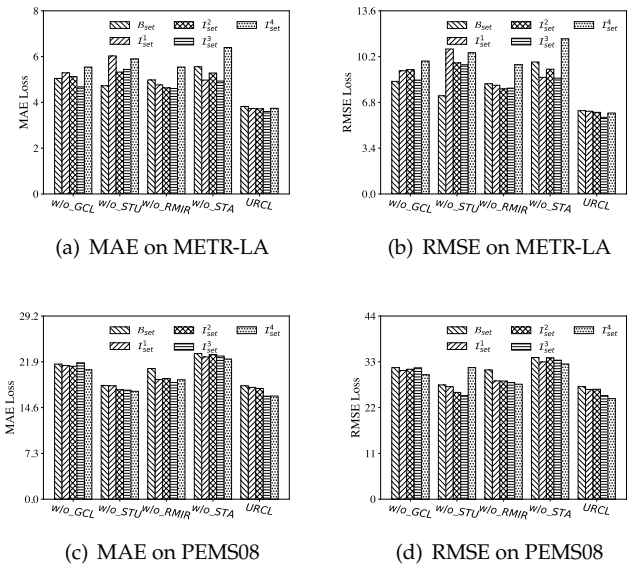
Table 2.2: Overall Accuracy on Four Datasets

Dataset		Metric	ARIMA	DCRNN	STGCN	MTGNN	AGCRN	STGODE	URCL
METR-LA	$\mathcal{B}_{set}$	MAE	4.99	4.01	3.92	4.01	3.83	7.39	**3.82**
		RMSE	8.82	6.22	7.98	6.89	6.33	11.42	**6.19**
	$\mathcal{I}^1_{set}$	MAE	5.87	10.55	9.99	5.83	6.78	7.11	**3.73**
		RMSE	9.77	14.41	14.77	9.71	12.42	11.62	**6.14**
	$\mathcal{I}^2_{set}$	MAE	5.88	7.59	3.83	4.33	4.30	7.35	**3.72**
		RMSE	10.49	8.44	5.63	6.89	7.59	11.77	**6.06**
	$\mathcal{I}^3_{set}$	MAE	4.88	4.77	4.96	3.74	4.32	6.45	**3.60**
		RMSE	10.49	6.81	7.00	5.91	7.59	10.17	**5.69**
	$\mathcal{I}^4_{set}$	MAE	5.00	4.13	3.89	4.31	4.32	7.40	**3.74**
		RMSE	10.99	6.25	**5.63**	6.94	7.53	11.78	6.01
PEMS-BAY	$\mathcal{B}_{set}$	MAE	1.98	1.30	1.31	1.39	1.25	4.89	**1.12**
		RMSE	3.85	2.24	2.28	2.53	2.51	7.93	**1.92**
	$\mathcal{I}^1_{set}$	MAE	3.88	1.11	2.57	1.30	1.14	4.66	**1.09**
		RMSE	4.98	1.87	5.58	2.22	2.23	7.61	**1.86**
	$\mathcal{I}^2_{set}$	MAE	3.67	1.30	2.20	1.47	1.74	2.44	**1.21**
		RMSE	4.89	2.34	4.55	2.53	2.24	3.69	**2.08**
	$\mathcal{I}^3_{set}$	MAE	3.24	1.21	3.33	1.20	1.62	5.31	**1.19**
		RMSE	6.77	2.11	7.06	2.13	2.01	8.64	**2.00**
	$\mathcal{I}^4_{set}$	MAE	5.92	1.22	3.49	1.17	1.73	4.84	**1.15**
		RMSE	10.22	2.00	7.53	2.10	2.05	7.84	**1.99**
PEMS04	$\mathcal{B}_{set}$	MAE	56.79	24.46	31.67	22.99	22.09	36.54	**19.77**
		RMSE	77.89	33.85	45.33	32.99	36.32	51.70	**31.26**
	$\mathcal{I}^1_{set}$	MAE	65.78	24.34	46.29	24.76	24.65	48.55	**21.56**
		RMSE	87.94	39.33	62.39	35.33	40.22	67.01	**34.10**
	$\mathcal{I}^2_{set}$	MAE	67.93	23.27	59.07	29.70	21.93	49.60	**21.61**
		RMSE	79.02	36.68	79.47	39.64	35.97	67.40	**34.04**
	$\mathcal{I}^3_{set}$	MAE	63.92	25.95	29.08	26.04	21.34	49.99	**20.97**
		RMSE	75.02	35.28	33.26	35.76	32.86	68.32	**32.12**
	$\mathcal{I}^4_{set}$	MAE	62.76	23.89	23.10	25.81	20.81	49.81	**20.48**
		RMSE	78.22	33.42	31.14	35.79	32.66	67.69	**32.20**
PEMS08	$\mathcal{B}_{set}$	MAE	45.98	23.07	18.33	20.70	23.03	28.58	**18.08**
		RMSE	67.28	32.24	27.68	30.38	38.44	38.54	**27.03**
	$\mathcal{I}^1_{set}$	MAE	55.22	20.35	18.35	20.66	19.47	36.55	**17.82**
		RMSE	75.33	27.92	**24.30**	29.82	31.74	46.04	26.37
	$\mathcal{I}^2_{set}$	MAE	49.33	22.15	39.32	20.27	17.45	36.06	**17.45**
		RMSE	59.89	29.63	57.34	28.97	27.35	45.25	**26.02**
	$\mathcal{I}^3_{set}$	MAE	58.99	19.94	44.22	22.68	17.70	36.12	**16.42**
		RMSE	65.79	27.04	63.01	30.24	27.56	45.04	**24.88**
	$\mathcal{I}^4_{set}$	MAE	62.89	19.87	27.21	18.92	18.35	36.05	**16.44**
		RMSE	71.36	27.91	37.19	26.99	28.80	45.30	**24.16**

datasets exceed those on the traffic flow datasets. This is because because much more traffic flow data is available than traffic speed data. Thus, the baselines using the replay-based training strategy with sufficient data can get comparable results.

- ARIMA, a traditional statistical method, performs worse than all methods except STGODE on METR-LA and PEMS-BAY and has the worst performance among all baselines on PEMS04 and PEMS08. This is because ARIMA only uses the time-series data of each region and ignores spatial dependencies. On METR-LA and PEMS-BAY, STGODE performs the worst since it is customized for speed prediction instead of flow prediction.
- As a popular spatio-temporal prediction model, AGCRN performs the best among all the baselines in most cases, due to its powerful ability to capture spatial and temporal correlations.

From the above observations, we can see that URCL can be applied on both traffic speed and traffic flow prediction tasks, which indicates generality across prediction tasks.



(a) MAE on METR-LA (b) RMSE on METR-LA

(c) MAE on PEMS08 (d) RMSE on PEMS08

Fig. 2.5: RMSE and MAE of URCL and Its Variants

Ablation Study

To gain insight into the effects of key aspects of URCL, including the STMixup mechanism, the RMIR sampling strategy, the data augmentation methods, and the GraphCL loss, we evaluate four URCL variants:

- ***w/o STMixup (w/o_STU).*** *w/o_STU* is the URCL model without the STMixup module. We directly concatenate the original observations and sampled observations from the replay buffer.
- ***w/o RMIR Sampling (w/o_RMIR).*** *w/o RMIR* replaces the RMIR sampling mechanism from URCL with a random sampling one.
- ***w/o STAugmentation (w/o_STA).*** *w/o_STA* is the URCL model without the STAugmentation.
- ***w/o GraphCL (w/o_GCL).*** *w/o_GCL* is the URCL model without the GraphCL loss.

Figure 2.5 shows the MAE and RMSE results on METR-LA and PEMS08. Regardless of the datasets, URCL always outperforms its counterparts without the STMixup mechanism, the RMIR sampling strategy, the data augmentation methods, or the GraphCL loss. These four components all improve the prediction accuracy of URCL, as removing any one of them increases the MAE and RMSE values on both base sets and incremental sets. Further, on both datasets, *w/o_STA* performs the worst among all variants in most cases. URCL outperforms *w/o_STA*, reducing the MAE and RMSE values by up to 41.5% and 47.8%, respectively, thus showing the benefit of spatio-temporal data augmentation. It is noteworthy that we include all the results in the technical report.

Chapter 3

LightTR+: A Lightweight Incremental Framework for Federated Trajectory Recovery

This chapter provides an overall introduction to Paper C [46]. Content is reused from the paper [46] in the most effective way.

1 Problem Motivation and Statement

Thanks to the rapid adoption and advancement of mobile sensing devices, massive amounts of trajectory data are being collected in a decentralized manner, enabling a wide range of trajectory-based applications [15, 20, 38], such as traffic prediction [43], destination prediction [64], and vehicle navigation [38]. However, in practice, trajectory data is often sampled at a low rate [88], resulting in incomplete trajectories (a.k.a. low-sampling-rate trajectories), which reduces the effectiveness of the above-mentioned applications due to the loss of detailed information and increased uncertainty. Thus, it is important to recover missing points in incomplete trajectories, a process known as *trajectory recovery*, to enable more effective utilization of low-sampling-rate trajectories.

Due to its significance, considerable research efforts have been dedicated to designing effective trajectory recovery models [2, 6, 53, 60, 76, 83]. However, existing methods assume that models are trained with centralized data collected from edge devices, leading to high collection and storage costs and failing to handle decentralized training data. Additionally, with increasing privacy

concerns, individuals may be unwilling to disclose their raw trajectories to untrusted data providers, worrying that their data can be misused by malicious third parties. Thus, decentralized trajectory processing, which enables privacy protection, has become a key issue in current trajectory-based applications to comply with licensing agreements and data access restrictions [43].

In this study, we aim to develop a novel FL-based trajectory recovery model that combines decentralized data processing and complex spatio-temporal dependency modeling. Developing such a model is a non-trivial task due to the following four challenges.

First, existing popular trajectory recovery methods often suffer from poor scalability, as deep learning-based models are often large, rendering both training and inference time-consuming and computationally expensive. This limitation is particularly problematic for resource-constrained edge computing devices, which are essential for decentralized computation. Second, during training in FL, multiple rounds of communication occur between the central server and all participating clients. Limited network bandwidth and a large number of participating clients can create a communication bottleneck, leading to increased latencies and reduced practicality. Third, mitigating both inter- and intra-domain interference in FL for trajectory recovery is particularly challenging. FL typically involves two phases: aggregation and local updating [13, 70]. During the aggregation phase, the central server collects and combines local parameters from active clients. his process can introduce potential inter-interference by other clients, where the global model becomes confused due to the heterogeneous (dissimilar) data from various clients. This inter-interference can hinder model convergence and degrade performance. In the local updating phase, each client optimizes its local model with continuously-generated private data, which can lead to overfitting to the current knowledge and forgetting of previous knowledge, called intra-domain interference, which results in catastrophic forgetting [19]. Finally, trajectory recovery often involves dealing with data arrives in real-time from various sources such as GPS devices and mobile applications. This makes it inherently a streaming challenge due to the continuous data generation, requiring models to adapt in real time.

To address these challenges, we propose LightTR+, a lightweight incremental framework for federated trajectory recovery based on horizontal federated learning (i.e., a server-clients architecture). The objective is to collaboratively train models by maintaining a shared global model on a central server while utilizing the data of each client in a decentralized fashion. Specifically, to avoid high memory consumption and limited scalability (*Challenge I*), we designed a local lightweight trajectory embedding (LTE) model for each client. The LTE model contains an embedding component and a stack of ST-blocks to learn effective spatio-temporal representations. To reduce communication costs and accelarate model convergence (*Challenge II*), we proposed a meta-knowledge enhanced local-global training module based on knowledge distillation. Before

federated training, we introduced a Teacher model (i.e., a meta-learner) to learn local meta-knowledge for each client using a portion of its local data. The local lightweight trajectory embedding model acts as the student model. During FL, the Teacher model guides the optimization of student models to learn better common features and achieve faster convergence. to alleviate intra-domain interference (*Challenge III*), we propose an intra-domain knowledge distillation module. This module leverages the previously trained local trajectory embedding model to transfer knowledge of old data to the current data for each client in the local updating phase. Additionally, the proposed meta-knowledge enhanced local-global training module can learn effective meta-knowledge to alleviate data heterogeneity across clients, thereby limiting inter-domain interference (*Challenge III*) in the aggregation phase. To address *Challenge IV*, we save the local trajectory model learned on previous local data and use it to constrain subsequent local updating with a distillation loss. When new data arrives, we distill the knowledge from the saved model to the newly updated model learned on the current data. Our goal is to enable the local model on each client to perform well on incoming data while preserving the knowledge learned from previous data as much as possible.

The major contributions are summarized as follows.

- To the best of our knowledge, this is the first study to systematically explore federated incremental learning on decentralized trajectory data, ensuring privacy protection for locally collected trajectory data. We propose a lightweight federated incremental framework entitled LighTR+ for trajectory recovery, which leverages knowledge distillation to alleviate catastrophic forgetting.
- To increase the scalability, we introduce a local trajectory preprocessing and light embedding module to capture effective spatio-temporal correlations of trajectories. To reduce the communication cost in FL, we design a meta-knowledge enhanced local-global training module by means of knowledge distillation.
- To alleviate intra-domain knowledge interference we propose an intra-domain knowledge distillation module. In addition, we employ a distillation loss to enable streaming learning
- We report on experiments using real data, providing insight into the effectiveness and efficiency of the proposed LightTR+ framework in alleviating catastrophic forgetting on decentralized trajectories.

2 Preliminaries

We proceed to present necessary preliminaries and then define the problem addressed.

2.1 Basic Concepts

Definition 7 (Road Network) *A road network is a directed graph $G = (V, E)$, where V is a road vertex set, and E is a road edge set. Each $v_i \in V$ denotes a road segment intersection or a road end, and each $e_{i,j} \in E$ denotes a directed road segment from v_i to v_j.*

Definition 8 (GPS Point) *A GPS point can be defined as a triple $p = \langle lat, lng, \gamma \rangle$, which captures the latitude lat and longitude lng of the GPS position including additional information γ, e.g., address.*

Definition 9 (Raw Incomplete Trajectory) *A raw incomplete trajectory τ is a sequence of n points, i.e., $\tau = \langle (p_1, t_1), (p_2, t_2), \cdots, (p_n, t_n) \rangle$, where t_i denotes the timestamp of p_i.*

Definition 10 (Sampling Rate) *A sampling rate ϵ is the time difference between two consecutively sampled points, which is usually determined by the sampling device.*

In practice, the sampling rate often changes near ϵ in most cases because of the inherent temporal bias of trajectory data. It is also worth noting that the collected trajectory points may not be precise due to GPS device measurement errors and GPS noises. Map matching is usually adopted to snap raw points onto the road network [8, 10, 52, 53]. In this study, we use a map-matching method based on deep hierarchical networks (DHN) [90] to convert a sequence of raw latitude/longitude coordinates for map matching.

Definition 11 (Map-matched ϵ-Sampling-Rate Trajectory) *A map-matched trajectory T with ϵ-sampling rate is a sequence of map-matched trajectory points, i.e.,*

$$T = \langle (\widetilde{p}_1, t_1), \cdots, (\widetilde{p}_i, t_i), \cdots, (\widetilde{p}_n, t_n) \rangle, \tag{3.1}$$

where $t_{i+1} - t_i = \epsilon$, $\widetilde{p}_i = \langle e, r \rangle$ is the map-matched (trajectory) point at timestamp t_i, e denotes the road segment, and r is the moving ratio. Next, r represents the ratio between the current moving distance (from the start node of e to the current node) and the length of road segment e, i.e., $r = \frac{dis(e.N_1, e.N_{cur})}{dis(e.N_1, e.N_2)}$, where $dis(a, b)$ denotes the distance between nodes a and b, $e.N_1$ denotes the start node of e, $e.N_{cur}$ denotes the current node of e, and $e.N_2$ denotes the end node of e.

As shown in Figure 2, each road segment e covers the start node $e.N_1$ and end node $e.N_2$, and records the distance between the two points. Next, the

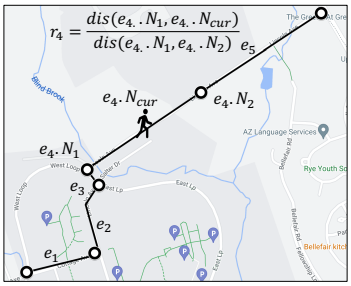


Fig. 3.1: Moving Ratio

moving ratio r will be determined by the moving distance (from $e.N_1$ to the current node $e.N_{cur}$) and the total length of e. For example, if the current node is located in the middle of e_4, we can obtain that $r = 0.5$.

Definition 12 (Incomplete Map-matched ϵ-Sampling-Rate Trajectory) *An incomplete map-matched ϵ-sampling-rate trajectory, i.e.,*

$$T_{icp} = \langle (\widetilde{p}_1, t_1), \cdots, (\bar{p}_j, t_j), \cdots, (\widetilde{p}_n, t_n) \rangle, \tag{3.2}$$

is composed of a sequence of map-matched points (e.g., $\widetilde{p}_i$) and missing points (e.g., $\hat{p}_j$).

In the rest of the paper, we will use the terms *map-matched point* (*map-matched ϵ-sampling-rate trajectory*) and *point* (*trajectory*) interchangeably when the context is clear.

Definition 13 (Platform Center) *A platform center C, e.g., the distributed data center of a company, keeps its map-matched trajectory dataset $\mathcal{T} = \{T_1, T_2, \cdots, T_l\}$, where T_i is the i-th map-matched trajectory and l is the length of $\mathcal{T}$. Note that we consider a platform as a client.*

Based on the above definitions, we formally define the studied problem as follows.

2.2 Problem Definition

Federated Trajectory Recovery. Given a server $\mathcal{S}$ and N clients with their local trajectory datasets $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \cdots, \mathcal{T}_N\}$, each dataset $\mathcal{T}_i$ $(1 \leq i \leq N)$ in client C_i is a set of incomplete map-matched trajectories, i.e., $\mathcal{T}_i = \left\{T^i_{icp,1}, \cdots, T^i_{icp,l}\right\}$. Our problem aims to learn a shared global function $F(\cdot)$, such that for any incomplete map-matched trajectory, their missing coordinates are recovered. Formally, for each incomplete trajectory T_{icp} in a platform center, we have

$$\overbrace{\langle \cdots, (\bar{p}_i, t_i), \cdots, (\widetilde{p}_j, t_j), \cdots \rangle}^{\text{Incomplete Trajectory}} \xrightarrow{F(\cdot)} \overbrace{\langle (\cdots, (\hat{p}_i, t_i), \cdots, (\widetilde{p}_j, t_j), \cdots \rangle}^{\text{Recovered Trajectory}}, \tag{3.3}$$

where $1 \leq i < j \leq n$.

3 Methodology

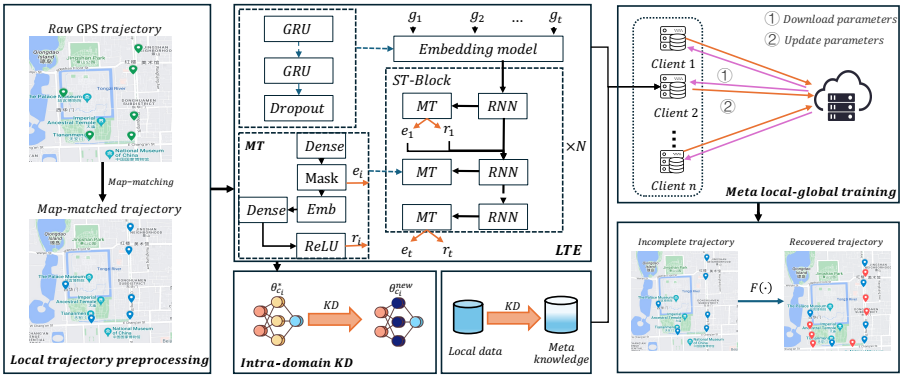


Fig. 3.2: LightTR+: Lightweight Trajectory Recovery Framework

In this section, we propose a federated trajectory recovery framework, entitled LightTR+, as shown in Figure 3.2. As shown in Figure 3.2, LightTR+ performs horizontal federated learning, i.e., a server-clients architecture, to collaboratively train local trajectory learning models under the orchestration of a central server, while keeping the local data privacy. LightTR+ can be divided into three modules: local trajectory preprocessing and light embedding, intra-domain knowledge distillation (KD), and meta-knowledge enhanced local-global training.

3.1 Local Trajectory Preprocessing and Light Embedding

In this section, we report the preprocessing of local trajectories and the proposed lightweight trajectory embedding module in a client. Note that the model for the server has the same trajectory learning model architecture, to be detailed in Section 3.1, as each clients.

Local Trajectory Preprocessing

Given a road network, we first map all collected trajectories into the corresponding road intersections by converting their GPS location into discrete

units, referring to the Hidden Markov Model (HMM) in DHN [90]. Specifically, given a low-sampling-rate trajectory τ, we obtain the corresponding map-matched trajectory T as follows:

$$\begin{aligned} \tau &= \langle (p_1, t_1), \cdots, (p_i, t_i), \cdots (p_n, t_n) \rangle \\ T &= \langle (g_1, t_1), \cdots, (g_i, t_i), \cdots (g_n, t_n) \rangle = HMM(\tau), \end{aligned} \tag{3.4}$$

where $g_i = (x_i, y_i, tid_i)$, $\forall 1 \leq i \leq n$ is the converted unit, and x_i and y_i represents the i-th grid cell. We extract $tid_i = \lfloor \frac{t_i - t_0}{\epsilon} \rfloor$ to guide the model to learn how many points should be recovered between two consecutive low-sampling-rate points.

Lightweight Trajectory Embedding

As shown in Figure 3.2, the Lightweight Trajectory Embedding (LTE) module is composed of an embedding model and stacked ST-blocks. The embedding model encodes the map-matched trajectory T into a single vector to capture its complex sequential dependencies, while the ST-blocks, containing a customized lightweight *ST-operator*, aim to predict the road segment e and moving ratio r of missing points, simultaneously.

We proceed to present the *embedding model* and the lightweight *ST-blocks*.

Embedding Model. Given a map-matched incomplete trajectory $T_{icp} = <(\widetilde{p_1}, t_1), \cdots, (\widetilde{p_i}, t_i), \cdots, (\widetilde{p_n}, t_n)>$, the embedding model aims to convert T into a single vector to capture the complex spatial and temporal correlations. It is important to model the sequential dependencies of trajectories, which enables accuracy [76]. We adopt the Gated Recurrent Unit (GRU) architecture for the embedding layer since GRUs bring the benefits of efficient long-term temporal dependency capture without performance decay. We input the map-matched raw trajectory T into the GRU to obtain its hidden features, which are then input into the ST-blocks. Specifically, we pass the input trajectory T along the time span $[1, 2, \cdots, n]$ through the GRU. For each time step t, the hidden features can be formulated as follows,

$$\begin{aligned} r_t &= \sigma(W_r \cdot [h_{t-1}, g_t] + b_r) \\ z_t &= \sigma(W_z \cdot [h_{t-1}, g_t] + b_z) \\ \tilde{h_t} &= tanh(W_h \cdot [r_i * h_{t-1}, g_t] + b_h) \\ h_t &= (1 - z_t) * h_{t-1} + z_i * \tilde{h_t}, \end{aligned} \tag{3.5}$$

where W represents the weight for respective gate neurons and b is the bias for the respective gate, and $[\cdot]$ denotes the feature concatenation.

For simplicity, the embedding model derives the hidden features h_t of the encoder for low-sampling-rate trajectories embedding:

$$h_t = embedding(h_{t-1}, s_{t-1}), \tag{3.6}$$

where the last state h_t is considered as the learned hidden features, which is also used as the input of ST-blocks.

ST-blocks. The learned hidden features h_t are then input into the stacked ST-blocks to obtain the data representations for incomplete trajectory recovery. As illustrated in Figure 3.2, to increase scalability and reduce computational costs, we design a lightweight *ST-operator* including an RNN layer, where each RNN cell is followed by a pure MLP-based multi-task (MT) model, which benefits from the low space and time complexities of MLPs. The RNN can only predict the numerical coordinates of trajectories; it cannot ensure that the predicted missing points fall onto the road network. Thus, the lightweight *ST-operator* contains a novel MT model to predict the road segment e and moving ratio r at the same time due to their high correlations, leveraging multi-task learning [51].

More specifically, we feed h_t into the RNN to obtain the high-dimensional hidden features h'_t, which capture the sequential dependencies. Next, h'_t is input into the MT model to predict the road segment e_t and moving ratio r_t simultaneously. The lightweight *ST-operator* is formulated as follows.

$$\begin{aligned} h'_t &= RNN(h_t) \\ e_t, r_t &= MT(h'_t) \end{aligned} \tag{3.7}$$

The left center in Figure 3.2 shows the detailed architecture of the MT model to predict e_t and r_t, simultaneously. Specifically, after obtaining hidden features through the RNN, we first use a dense (MLP) layer with a constraint mask layer (Mask) and a road segment embedding layer (Emb) to predict the road segment e_t, which is formulated as follows.

$$\begin{aligned} h_{t,d} &= Dense(h'_t, W_d) = W_d \cdot h'_t + b_d \\ e^t &= Mask(h_{t,d}) \\ h_{t,e} &= Emb(h_{t,d}, e^t) = ReLU(h_{t,d} + RNN(e^t)) \\ r_t &= ReLU(Dense([h_{t,e}, e^t], W_r)) = W_r \cdot [h_{t,e}, e^t] + b_r, \end{aligned} \tag{3.8}$$

where W and b denote the trainable parameters and bias, respectively.

Then, e_t together with h'_t are concatenated to go through the dense layer with an activation function (i.e., ReLU) to predict the moving ratio r_t. Next, e_t and r_t affect e_{t+1} and r_{t+1}. For example, people may be more willing to visit places that are close to their current locations. We also input e_t and r_t into the RNN. When inferring the missing points between two consecutive points of a low-sampling-rate trajectory, the hidden state h_t of our ST-blocks can be represented as follows.

$$h_t = STBlocks(h_{t-1}, e_{t-1}, r_{t-1}), \tag{3.9}$$

where e_{t-1} and r_{t-1} represent the road segment embedding of the predicted road segment and its moving ratio at timestamp $t-1$, respectively.

Constraint Mask Layer. In order to reduce the training complexities and achieve fine-grained trajectory recovery, we employ a constraint mask layer [53]. We use an exponential function (shown in Eq. 3.10) to capture the influence of distance.

$$c_i = exp(\frac{-dist^2(p_i, \widetilde{p_i})}{\gamma}), \tag{3.10}$$

where $dist(p_i, \widetilde{p_i})$ denotes the Euclidean distance between the original point p_i and its map-matched point $\widetilde{p_i}$ on the corresponding road segment e and γ is a parameter related to the road network ($\gamma = 125$ in our work). Note that we only consider points in low-sampling-rate trajectories that are not far away from road segments. If the distance between a road segment e and point p_i is more than 50 meters, we set $\omega(e, p_i)$ to 0. Finally, we combine Eq. 3.10 with softmax as the constraint mask layer, expressed as follows.

$$P(e_i|h_i) = \frac{exp(h_{t,d}^T \cdot w_c) \odot c_i}{\sum_{c' \in C} exp(h_{t,d}^T \cdot w_{c'}) \odot c_i}, \tag{3.11}$$

where w_c is a trainable parameter matrix. We then use $argmax$ to get the final prediction of road segment e^t.

Intra-Domain Knowledge Distillation

To alleviate intra-domain interference and catastrophic forgetting, we propose an intra-domain knowledge distillation module that enables effective knowledge transfer from a model learned on previous trajectory data to an updated model learned on incoming trajectory data. To enable streaming learning, we save the local trajectory model learned on previous local data and use it to constrain subsequent local updating with a distillation loss.

In particular, we perform two training steps for each client. First, we train a local model $\theta_{c_i}^*$ on client c_i's private trajectory data by means of the proposed local trajectory embedding model. Second, when new data arrives, we finetune the current local model $\theta_{c_i}^*$ with the new data to obtain a new local model $\theta_{c_i}^{new}$. We apply a distillation loss $L_{distill}$ between $\theta_{c_i}^{new}$ and $\theta_{c_i}^*$ to facilitate effective knowledge transfer from the previously trained data to the new data in order to avoid intra-domain interference. The distillation loss is designed based on knowledge distillation, where we consider $\theta_{c_i}^*$ and $\theta_{c_i}^{new}$ as the teacher and a student model, respectively. This is formulated as follows.

$$L_{distill} = softmax(\theta_{c_i}^*) log \frac{softmax(\theta_{c_i}^*)}{softmax(\theta_{c_i}^{new})} \tag{3.12}$$

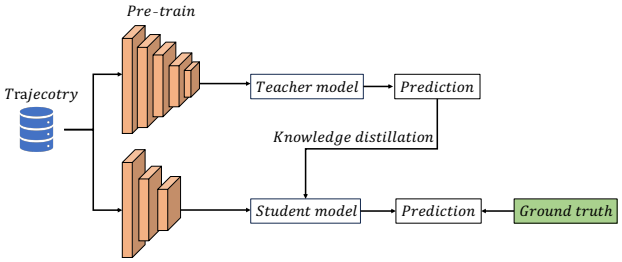


Fig. 3.3: The Process of Knowledge Distillation

3.2 Meta-knowledge Enhanced Local-Global Training

To contend with inter-domain interference and reduce the communication cost, we propose a meta-knowledge enhanced local-global training (or meta local-global training for short) module for model aggregation by means of knowledge distillation. Figure 3.3 shows the process of knowledge distillation, which includes a teacher model and a student model. Specifically, we first pre-train a teacher model, also called a meta-learner, to learn meta-knowledge for each client with a subset of local trajectories (see Algorithm 1). During local training, each client first downloads parameters from the server, and the learned meta-knowledge is used to guide the local model (i.e., lightweight trajectory embedding) learning. We consider the local model as the student model. Finally, the server aggregates parameters to update the global model with a subset of clients. In this study, we use the lightweight trajectory embedding model as the student and teacher model.

In federated trajectory recovery, we have N clients, denoted as $\{C_1, C_2, \cdots, C_N\}$, with their own datasets $\{\mathcal{T}_1, \mathcal{T}_2, \cdots, \mathcal{T}_N\}$. Each dataset $\mathcal{T}_i$ consists of three parts, a training, validation, and test dataset, denoted as $\mathcal{T}_i^{train}$, $\mathcal{T}_i^{valid}$, and $\mathcal{T}_i^{test}$, respectively. We aim to learn a good trajectory recovery model $f_i(\cdot)$ for each client on its local dataset $\mathcal{T}_i$:

$$min \frac{1}{N} \sum_{i=1}^{N} \frac{1}{n_i^{test}} \sum_{j=1}^{n_i^{test}} L_{total}(f_i(T_{icp}^j), T_{ground}^j), \tag{3.13}$$

where L_{total} is the overall loss function, n_i^{test} is the length of $\mathcal{T}_i^{test}$, and T_{ground} is the ground truth of complete trajectory.

We proceed to introduce the local trajectory recovery objective function and meta-knowledge enhanced local-global training.

Local Trajectory Recovery Objective

As we aim to predict road segment and moving ratio simultaneously, we combine the cross-entropy loss $L_1(\theta)$ and mean squared error $L_2(\theta)$ as the local

loss function $L_{local}(\theta)$ for trajectory recovery. $L_1(\theta)$ is used for road segment prediction, while we use $L_2(\theta)$ for moving ratio prediction.

$$L_{local}(\theta) = L_1(\theta) + \mu L_2(\theta), \tag{3.14}$$

where μ is a tunable parameter to balance the trade-off linearly between the road segment and moving ratio prediction. To enable streaming learning, we save the local trajectory model learned on previous local data, which is adopted to constrain the subsequent local updating with a distillation loss. Specifically, when new trajectory data arrives, we employ an additional distillation loss $L_{distill}$ to obtain a local loss function $L_{local}(\theta)$, formulated as follows.

$$L_{local}(\theta) = L_1(\theta) + \mu L_2(\theta) + L_{distill}, \tag{3.15}$$

where $L_{distill}$ is introduced to alleviate the catastrophic forgetting incurred by intra-domain interference.

The cross-entropy loss is formulated as follows.

$$\begin{aligned} L_1(\theta) = - \sum_{(T,\tilde{T}\epsilon\mathcal{T})} \sum_{j=1}^{|\tilde{T}|} \sum_{l=1}^{L} 1a_j \cdot e_l log(R_\theta(\hat{a}_j e = e_l | d_1 : j-1)) \\ s.t. d_{j-1} = (T, \tilde{T}_{1:j-1}), \end{aligned} \tag{3.16}$$

where T and $\tilde{T}$ represent incompleted trajectories and recovered completed trajectories, respectively, L is the number of road segments, $a_j e$ is the ground truth of road segment, $\hat{a}_j$ is the prediction, and R_θ represents the lightweight trajectory embedding module. Next, D is the training data.

The mean squared error is also employed to predict the movement ratio, which can be formulated as:

$$\begin{aligned} L_2(\theta) = - \sum_{(T,\tilde{T}\epsilon\mathcal{T})} \sum_{j=1}^{|\tilde{T}|} (a_j r - R_\theta(d_{j-1})^2) \\ s.t. \, d_{j-1} = (T, \tilde{T}_{1:j-1}), \end{aligned} \tag{3.17}$$

where $a_j r$ is the real moving ratio.

Meta-knowledge Enhanced Local-global Training

To achieve faster convergence and reduce communication costs, we propose a meta-knowledge enhanced local-global training module that uses knowledge distillation. We first randomly select a subset from the local dataset for each client and pre-train a meta-learner (a teacher model) to accumulate meta-knowledge across clients. By combining this with knowledge distillation, the learned meta-knowledge can alleviate data heterogeneities, thus mitigating

Algorithm 1 Teacher model training

Input: subsets of Dataset $\{\mathcal{T}_i\}_{i=1}^{N}$ at N clients, λ_0, l_t
Output: A common teacher model f_{tea}
1: $\lambda \leftarrow \lambda_0$;
2: Train local teacher model f_i using L_{local} with $\mathcal{T}_i^{train}$ at each client C_i;
3: Send the current model f_i to the next client C_{i+1};
4: $acc_{i+1}^{vaild} \leftarrow f_{i+1}(\mathcal{T}_{i+1}^{valid})$;
5: **if** $acc_{i+1}^{valid} > l_t$ **then**
6: Train f_{i+1} with $\mathcal{T}_{i+1}$ using Eq.3.19;
7: **else**
8: $f_i \leftarrow f_{i+1}$
9: Train f_{i+1} with $\mathcal{T}_{i+1}$ using Eq.3.19;
10: **end if**
11: Repeat steps 3 – 10 until convergence;
12: $f_{tea} \leftarrow f_N$;
13: **Return** f_{tea}

inter-domain interference across clients. Each client trains a local model after downloading parameters from the server with meta-knowledge for guiding the local training. The server obtains a global model by aggregating parameters from the participating clients.

Algorithm 1 specifies the process of teacher model training, which is used by each clients. Here, we train all clients sequentially in a cyclic way, transfering previous knowledge to the next. The previously learned knowledge is preserved if it is useful for the current client training. Otherwise, it is discarded. This alleviates data heterogeneity across clients. We fix $\lambda = \lambda_0$ to preserve suficient common knowledge (line 1). Threshold l_t is used to determine whether to completely preserve the previously learned knowledge (lines 2–10). When accuracy $acc_{i+1}^{valid} < l_t$, too little information is contained in the current client C_{i+1}. We directly initialize the current model with the previous one to fully leverage previous knowledge, and thus handle the data heterogeneity.

We employ the proposed lightweight trajectory embedding model as the teacher model f_{tea} to perform the trajectory missing points prediction task with incomplete trajectories. The teacher model is then used to guide the training of the local model f_{stu} via knowledge distillation. The knowledge distillation is formulated as follows.

$$L_{dist}(f_{tea}, f_{stu}; T_{icp}) = ||f_{tea}(T_{icp}) - f_{stu}(T_{icp})||_2^2, \tag{3.18}$$

where $||\cdot||_2$ is the $L2\text{-}norm$.

Through knowledge distillation, we can make good use of meta-knowledge to guide the local training for each client, which alleviates data heterogeneities

across clients. Therefore, the total loss L_{total} function for each client C_i is:

$$L_{total} = \frac{1}{|\mathcal{T}^{train}|} \sum_{(T_{icp}, T_{ground}) \epsilon \mathcal{T}_i^{train}} L_{local}(R_\theta(T_{icp}), T_{ground}) + \lambda L_{dist}(f_{tea}, R_\theta; T_{icp}), \tag{3.19}$$

where λ is a trade-off knowledge transfer and focusing on the current data, and L_{local} is the trajectory recovery loss.

Algorithm 2 Meta-Knowledge Enhanced Local Training

Input: Datasets $\{\mathcal{T}_i\}_{i=1}^N$ in N clients, λ_0, l_t, teacher model f_{tea}, training epochs $epoch_{max}$

Output: Local Trajectory Recovery models $\{R_\theta^i\}_{i=1}^N$

1: $\lambda \leftarrow 0$;
2: $c_n \leftarrow 0$;
3: **while** $c_n < N$ **do**
4: $\quad acc_i^{train} \leftarrow R_\theta^i(\mathcal{T}_i^{train})$;
5: $\quad$ Train R_θ^i using Eq. 3.19;
6: $\quad acc_{i,tea}^{valid} \leftarrow f_{tea}(\mathcal{T}_i^{valid})$;
7: $\quad acc_i^{valid} \leftarrow R_\theta^i(\mathcal{T}_i^{valid})$;
8: $\quad$ **if** $acc_{i,tea}^{valid} \leq acc_i^{valid}$ and $acc_i^{valid} < l_t$ **then**
9: $\quad\quad \lambda \leftarrow 0$;
10: $\quad$ **else**
11: $\quad\quad \lambda \leftarrow \lambda_0 \cdot 10^{min(1,(acc_{i,tea}^{valid} - acc_i^{valid})*5)-1}$;
12: $\quad$ **end if**
13: **end while**
14: Repeat steps 3 – 15 in $epoch_{max}$ iterations;
15: **Return** $\{R_\theta^i\}_{i=1}^N$

Algorithm 2 illustrates the process of meta-knowledge enhanced local training (i.e., knowledge distillation) to train a good local trajectory recovery model for upcoming central aggregation. Specifically, we first set λ to 0, which means no knowledge is learned from the teacher model (line 1). For each client, we apply gradient descent to update its local model R_{θ_i} with local training data using Eq. 3.19 (lines 3–5). When the accuracy $acc_{i,tea}^{valid}$ of the teacher model is below acc_i^{valid} of the current local model on local validation data, we keep λ at 0, meaning that there is no guidance by the teacher model as it has less knowledge of the current client, and we do not want to rely on it. Otherwise, we dynamically update λ as follows.

$$\lambda \leftarrow \lambda_0 \cdot 10^{min(1,(acc_{i,tea}^{valid} - acc_i^{valid})*5)-1}, \tag{3.20}$$

Eq. 3.20 shows that the better the teacher model performs, the larger the value of λ is, which means that the local model learns more knowledge from the

Algorithm 3 Local-global Parameter Updating

Input: communication rounds R, initial global model of server θ_s^0, number of clients N, learning rate α, local training epochs E
Output: final global model of server θ_s

1: **for** each round r in $[1, 2, \cdots, R]$ **do**
2: Randomly select $C \in [1, N]$ clients from N clients;
3: **for** each client $c_i \in C$ **do**
4: The server transmits θ_s^{r-1} to client c_i;
5: **for** each local epoch e in $[1, 2, \cdots, E]$ **do**
6: $\theta_{c_i}^r \leftarrow argmin_\theta L_{total}(\theta_{c_i}^r)$;
7: $\nabla L_{total}(\theta_{c_i}^r) \leftarrow \nabla L_{total}(\theta_{c_i}^{r-1}) - \alpha(\theta_{c_i}^r - \theta_{c_i}^{r-1})$;
8: **end for**
9: $\theta_{c_i} \leftarrow \theta_{c_i}^r$;
10: **end for**
11: $\theta_s \leftarrow \sum_{c_i \in C} \frac{1}{C} \theta_{c_i}$;
12: **end for**
return θ_s

teacher model. After several iterations of model updating, we obtain N local models for the preceding central aggregation (lines 5–15).

Algorithm 3 shows the process of local-global parameter updating. At each communication round r, we randomly select a subset of N clients C, which further reduces communication costs. The server transfer its current model θ_s^{r-1} to selected clients (lines 1–4). Each client performs meta-knowledge enhanced local training according to Algorithm 2, which optimizes a local empirical risk objective, i.e., Eq. 3.19 (lines 5–10). Next, the selected clients upload their local model parameters θ_{c_i} to the server. Finally, the server aggregates the uploaded parameters to update the global model θ_s until it converges to a stationary point θ_s (line 11).

4 Experimental Evaluation

4.1 Experimental Setup

Datasets

The experiments are carried out on two real-world public trajectory datasets: Tdrive and Geolife.

- **Tdrive.** The Tdrive dataset contains taxi-trip-based trajectories in Beijing from February 2, 2008, to August 2, 2008, including roughly 15 million trajectory points.

- **Geolife.** The Geolife dataset contains 17621 taxi GPS trajectories in Asia from April 2007 to August 2012, including 182 taxi drivers.

Evaluation Metrics

We aim to recover low-sampling-rate trajectories in free space to high-sampling-rate trajectories mapped onto the road network. Thus, both accuracy of road segment recovery and distance error of location inference are adopted to compare the performance of our model and baselines.

Recall & Precision. We use recall and precision to evaluate the performance of route recovery by comparing the recovered road segments P_R to the corresponding ground truths G. Recall and precision are defined as follows.

$$Recall = \frac{|P_R \cap G|}{|G|}, Precision = \frac{|P_R \cap G|}{|P_R|} \tag{3.21}$$

MAE & RMSE. The distance measures, i.e., Mean Absolute Error (MAE) and Root Mean Square Error (RMSE), are used to evaluate the point recovery performance. Note that we calculate the distance error based on the road network by updating the Earth distance to road network constrained distance. The smaller the MAE and RMSE are, the better the models perform. MAE and RMSE are defined as follows.

$$\begin{aligned} MAE &= \frac{1}{m}\sum_{j=1}^{m}|dis(g_j, \hat{g}_j)|, \\ RMSE &= \sqrt{\frac{1}{m}\sum_{j=1}^{m}(|dis(g_j, \hat{g}_j)|)^2} \\ s.t. dis(g_j, \hat{g}_j) &= min(rn_{dis}(g_j, \hat{g}_j), rn_{dis}(\hat{g}_j, g_j)), \end{aligned} \tag{3.22}$$

where g_j is the ground truth location, $\hat{g}_j$ is the predicted map-matched trajectory point, and $rn_{dis}(g_j, \hat{g}_j)$ is the length of the shortest path between the prediction and the ground truth. Next, we use $min(rn_{dis}(g_j, \hat{g}_j), rn_{dis}(\hat{g}_j, g_j))$ as the final error since the road network is a directed graph, where the distance from g_i to g_j may not be equal to the distance from g_j to g_i.

Baseline

We compared the proposed LightTR+ with the following baselines. To the best of our knowledge, there is currently no FL-based method for discrete trajectory recovery. To enable fair comparisons, we transfer existing centralized baselines for trajectory recovery to their federated versions by combining them with FedAvg [41].

Table 3.1: Performance of Training on Streaming Data on Two Datasets (Keep Ratio = 12.5%)

Dataset	Method	M1 → Geolife		M1 → Tdrive		M2 → Geolife		M2 → Tdrive	
		Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision
Geolife	**MTrajRec+FL**	0.587	0.611	0.087	0.096	0.323	0.344	0.187	0.231
	LightTR	0.677	0.743	0.095	0.112	0.455	0.472	0.228	0.285
	LightTR+	0.695	0.724	0.108	0.134	0.511	0.546	0.311	0.437
TDrive	**MTrajRec+FL**	0.092	0.112	0.510	0.544	0.336	0.375	0.208	0.221
	LightTR	0.112	0.154	0.611	0.668	0.377	0.436	0.255	0.337
	LightTR+	0.146	0.114	0.608	0.661	0.522	0.612	0.410	0.487

- **FC+FL.** The FC+FL method combines horizontal FL with stacked fully connected layers (FC) [30] for trajectory recovery.
- **RNN+FL.** The RNN+FL method is a decentralized trajectory recovery model, where stacked RNNs are integrated with horizontal FL to collaboratively learn trajectory representations.
- **MTrajRec+FL.** The MTrajRec+FC devises a decentralized trajectory recovery model, where the MTrajRec [53] is employed as the local model.
- **RNTrajRec+FL.** The RNTrajRec+FL method is a horizontal FL trajectory recovery model, where we set the local model to be RNTrajRec [17], which applies a graph neural network to learn rich trajectory representations.
- **LightTR.** The LightTR method is a state-of-the-art federated trajectory recovery model with a lightweight network, which we proposed in our preliminary study [39].

4.2 Experimental Results

Performance on Training on Streaming Data

To evaluate the performance of training on streaming data, two representative baselines, MTrajRec+FL and LightTR, are compared with LightTR+. We first train an initial model M1 on either Geolife or Tdrive and then fine-tune M1 with the other dataset to obtain an updated model M2. The results are shown in Table 3.1, where M1 (or M2) → Geolife (or Tdrive) means that applying M1 (or M2) on Geolife (or Tdrive) for testing. The overall best performance is marked in bold. As the performance on both datasets exhibits a similar trend, for brevity, we only show the observations when M1 is trained on Geolife.

We first apply M1 on Geolife (M1 → Geolife) and Tdrive (M1 → Tdrive), simultaneously. LightTR+ consistently performs better than the two baselines. MTrajRec+FL and LightTR offer acceptable Recall and Precision on Geolife, but their performance deteriorates on Tdrive. LightTR+ performs better than the two baselines on Tdrive in most cases. This may be because the proposed

Table 3.2: Overall Performance Comparison on Two Datasets

Datasets	Baseline	6.25%				12.5%				25%			
		Recall	Precision	MAE	RMSE	Recall	Precision	MAE	RMSE	Recall	Precision	MAE	RMSE
Geolife	**FC+FL**	0.212	0.219	0.755	1.015	0.283	0.215	0.728	1.007	0.315	0.294	0.685	0.807
	RNN+FL	0.392	0.417	0.630	0.776	0.397	0.394	0.628	0.741	0.456	0.438	0.561	0.659
	MTrajRec+FL	0.595	0.601	0.462	0.634	0.599	0.623	0.455	0.636	0.634	0.673	0.410	0.546
	RNTrajRec+FL	0.594	0.618	0.455	0.617	0.631	0.655	0.431	0.611	0.669	0.702	0.398	0.501
	LightTR	0.674	0.679	0.375	0.464	0.724	0.748	0.335	0.432	0.758	0.739	0.323	0.435
	LightTR+	0.683	0.711	0.366	0.471	0.701	0.722	0.354	0.449	0.705	0.728	0.336	0.424
Tdrive	**FC+FL**	0.211	0.198	0.753	1.001	0.283	0.232	0.723	1.022	0.276	0.270	0.662	0.804
	RNN+FL	0.358	0.355	0.584	0.750	0.392	0.396	0.547	0.682	0.456	0.464	0.536	0.664
	MTrajRec+FL	0.512	0.554	0.532	0.697	0.543	0.587	0.486	0.628	0.633	0.685	0.452	0.530
	RNTrajRec+FL	0.568	0.611	0.488	0.651	0.598	0.643	0.447	0.585	0.620	0.705	0.384	0.473
	LightTR	0.624	0.674	0.395	0.473	0.667	0.707	0.366	0.452	0.708	0.732	0.332	0.446
	LightTR+	0.611	0.677	0.402	0.497	0.671	0.698	0.365	0.448	0.687	0.701	0.355	0.473

intro-domain knowledge distillation module enables more generalized feature learning.

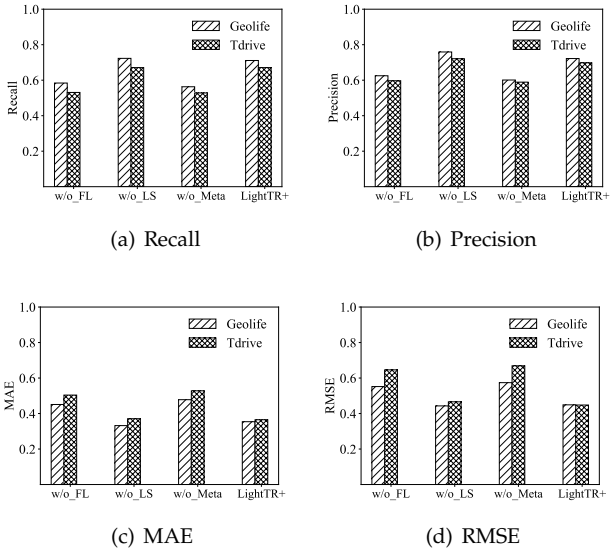
M2 is obtained by fine-tuning M1 with Tdrive. After fine-tuning, we apply the updated model M2 on Geolife (M2 → Geolife) and Tdrive (M2 → Tdrive) simultaneously for testing. We observe that LightTR+ achieves the best results among the methods, which is evidence that LightTR+ can perform effective feature learning on new data while preserving old knowledge. This is because of the intra-domain knowledge distillation module, which avoids catastrophic forgetting by enabling effective knowledge transfer from previously learned data to current data. Simple fine-tuning based methods are unable to handle streaming data, as they fail to overcome both inter- and intra-domain interference. LightTR+ achieves relatively stable performance on both Geolife and Tdrive, demonstrating its strength over baselines. In addition, both LightTR and LightTR+ consistently outperform MTrajRec+FL due to the powerful representation learning capability of the proposed *ST-operator*.

Baseline Comparison

Table 3.2 shows the performance comparison among different methods on the two datasets. The observations are as follows.

- LightTR+ and LightTR achieve comparable results in most cases on the two datasets with different keep ratios (i.e., 6.25%, 12.5%, and 25%), and they outperform all the baselines. In addition, LightTR+ performs better than LightTR by up to 4.7% in terms of precision on Geolife when the keep ratio is 6.25%. Although LightTR outperforms LightTR+ on Tdrive with a 25% keep ratio, LightTR+ can avoid catastrophic forgetting when it is applied to streaming trajectories, showing its superiority in incremental trajectory recovery scenarios.

- LightTR performs better than the best among the baselines by up to 14.7% and 13.19% with keep ratio 12.5% in terms of Recall and Precision, respectively, on Geolife. This is because of the powerful feature learning



(a) Recall (b) Precision

(c) MAE (d) RMSE

Fig. 3.4: Performance of LightTR and Its Variants on Both Datasets (keep ratio = 12.5%).

capability of the proposed light *ST-operator*. Moreover, the learned meta-knowledge alleviates the data heterogeneity across clients, which further improves the model performance. We also observe that the performance improvements obtained by LightTR on Geolife exceed those on Tdrive since the Geolife data are more sufficient than Tdrive with more useful knowledge.

- All RNN-based methods (i.e., RNN+FL, MTrajRec+FL, and RNTrajRec+FL), which are able to learn temporal dependencies of trajectories, perform better than FC+FL on both Tdrive and Geolife. It shows the importance of effective temporal dependency capturing for trajectory embedding. FC+FL performs the worst on both datasets, which demonstrates that simply stacked FC layers are too shallow to capture the complex temporal correlations and thus shows the necessity of the newly proposed lightweight *ST-operator*.
- RNN+FL, MTrajRec+FL, and RNTrajRec+FL can capture the temporal dependencies, RNTrajRec+FL performs the best. This is because RNTrajRec is capable of learning more effective trajectory embeddings benefiting from the graph model, which incorporates the embedding of each road segment.

The above observations indicate that LightTR is more effective than the existing FL-based methods.

Ablation Study

To gain insight into the effects of key aspects of LightTR+, We evaluate three LightTR+ variants.

- **w/o horizontal FL (w/o_FL).** w/o_FL removes the central server of LightTR+, where each client trains their model locally and all clients exchange their local model to each other.
- **w/o Lightweight *ST-operator* (w/o_LS).** w/o_LS replace the Lightweight *ST-operator* with MTrajRec to learn local trajectory embeddings for each client.
- **w/o meta-knowledge enhanced local-global training module (w/o_Meta).** w/o_Meta replace the meta-knowledge enhanced local-global training module with FedAvg [41].

To assess whether the components in LightTR+ all contribute to the performance of trajectory recovery, we compare LightTR+ with its variants. The results are shown in Figure 3.4. The observations are as follows.

- Regardless of the datasets, LightTR+ always performs better than its counterparts without horizontal FL, lightweight *ST-operator*, and meta-knowledge enhanced local-global training module. It shows that these components are all useful for trajectory recovery.
- LightTR+ performs slightly better than w/o_LS. This shows the powerful feature learning capability of the proposed lightweight *ST-operator*, even compared with MTrajRec, a recent state-of-the-art trajectory recovery model. However, the complex model always has high space and time complexities, which incurs poor scalabilities and more computation resource requirements.
- w/o_Meta performs the worst among all variants, which shows the importance of the meta-knowledge enhanced local-global training module. It also offers evidence that the meta-knowledge enhanced local-global training module is capable of alleviating the data heterogeneities (intra-domain interference) across clients.

Chapter 4

Federated Trajectory Similarity Learning with Privacy-Preserving Clustering

This chapter provides an overall introduction to Paper D [44]. Content is reused from the paper [44] in the most effective way.

1 Problem Motivation and Statement

With the proliferation of edge computing and devices enabling the capture of the movements of objects (e.g., vehicles and people), massive volumes of trajectory data are being collected in a decentralized fashion. A fundamental aspect of trajectory-based applications is trajectory similarity learning. The left part of Figure 4.1 illustrates trajectory similarity, where we consider T_2 and T_3 as similar trajectories due to their similar characteristics (e.g., origin, destination, and timestamp t_i). In addition, the distance (e.g., Euclidean distance) between T_2 and T_3 is smaller than that between T_2 (or T_3) and T_1.

Due to its significance, substantial research has been devoted to inventing effective trajectory similarity models [3, 11, 25, 29, 78]. In real-world scenarios, trajectories are often generated at edge devices that are distributed across cities. Yet, most existing trajectory similarity learning models [29, 33] assume centralized training data and fail to take into account the decentralized setting. We aim to invent a new decentralized model that can utilize decentralized data and can learn complex spatio-temporal semantics of trajectories.

However, it is non-trivial to develop such a model, due to two main challenges. First, in order to achieve effective trajectory learning, existing methods

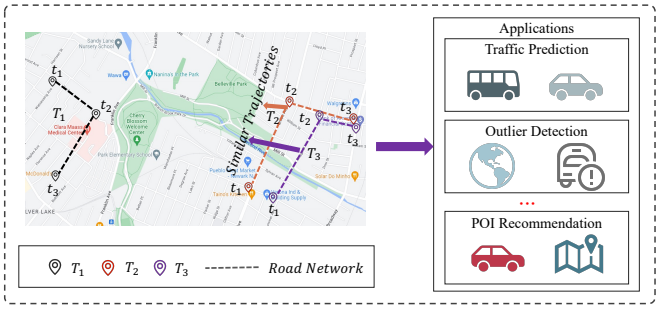


Fig. 4.1: Illustration of Trajectory Similarity

require aggregation of all available trajectories, which may reveal real sensitive information that may be exploited by a malicious third party. Although recent studies in Federated Learning (FL) provide solutions for decentralized data training on multiple devices and considering privacy, to our knowledge, no FL models exist that can capture spatial dependencies and temporal correlations effectively in terms of trajectory similarity computation. Second, the distributions of the trajectories that are collected across different edge devices are usually not independent and identically distributed (Non-IID) rather, they have been found to be remarkably heterogeneous [49, 63]. The distribution drift across different local datasets makes it difficult to achieve an optimal global model by aggregating local models.

To address these challenges, we propose F-TSL, a federated trajectory similarity learning framework. F-TSL is designed to enable horizontal federated learning (i.e., server-clients), where many clients collaboratively train a global model using local training data under the orchestration of a central server. F-TSL encompasses three major modules: local trajectory preprocessing and learning, privacy-preserving clustering, and server-side aggregation. To enable federated learning that supports the unique properties of trajectories, notably spatio-temporal correlations, we design a local trajectory preprocessing and learning module with a well-customized local trajectory learning model for use by clients. Next, the privacy-preserving clustering module aggregates similar clients together to contend with trajectory heterogeneity by means of a differential privacy method (i.e., Laplacian [22]) to avoid information leakage during clustering.

The major contributions are summarized as follows.

- To the best of our knowledge, this is the first systematic study of federated trajectory similarity learning. We propose F-TSL, a federated similarity learning framework for decentralized trajectory similarity learning, which offers privacy protection of locally collected trajectories.

- We introduce a local trajectory preprocessing and learning module to capture effectively spatio-temporal correlations of trajectories by means of a well-customized local trajectory learning model.
- To contend with trajectory heterogeneity, a novel privacy-agnostic clustering mechanism is proposed to facilitate more consistent model updates between clients and the server. In addition, we introduce an alignment loss to minimize the differences between local models and the global model.
- We report on experiments on real data, offering evidence of the effectiveness of the new framework and its components.

2 Preliminaries

2.1 Basic Concepts

Definition 14 (Road network) *A road network is modeled as a directed graph $G = (V, E)$, where V is a road vertex set, and E is an edge set. Each vertex $v_i \in V$ represents a road intersection or a road end, and each edge $e_{i,j} \in E$ indicates a directed road segment from v_i to v_j.*

Definition 15 (Point) *A point is a triple $p = \langle lat, lon, \gamma \rangle$ that records a latitude lat and longitude lon including additional information γ, e.g., address.*

Definition 16 (Raw Trajectory) *A trajectory τ is a sequence of points at n discrete timestamps, i.e.,*

$$\tau = \langle (\widetilde{p}_1, t_1), (\widetilde{p}_2, t_2), \cdots, (\widetilde{p}_n, t_n) \rangle, \tag{4.1}$$

where $\widetilde{p}_i$ denotes a geo-location (point) and t_i is the corresponding timestamp.

In practice, collected trajectory points may not be accurate due to GPS device measurement noises or errors. Map-matching is usually adopted to project raw points onto an underlying road network [24]. We use the Incremental Map-Matching [8] method to convert a sequence of raw coordinates to coordinates in the road network that then become a sequence of road-network vertices that encode a connected sequence of edges.

Definition 17 (Map-matched Trajectory) *Given a raw trajectory τ and an underlying road network $G = (V, E)$, the trajectory T obtained by map-matching of τ is a sequence of n vertices in G, i.e.,*

$$T = \langle (p_1, t_1), (p_2, t_2), \cdots, (p_n, t_n) \rangle, \tag{4.2}$$

where $p_i \in V$ denotes a vertex in G and t_i represents the corresponding timestamp.

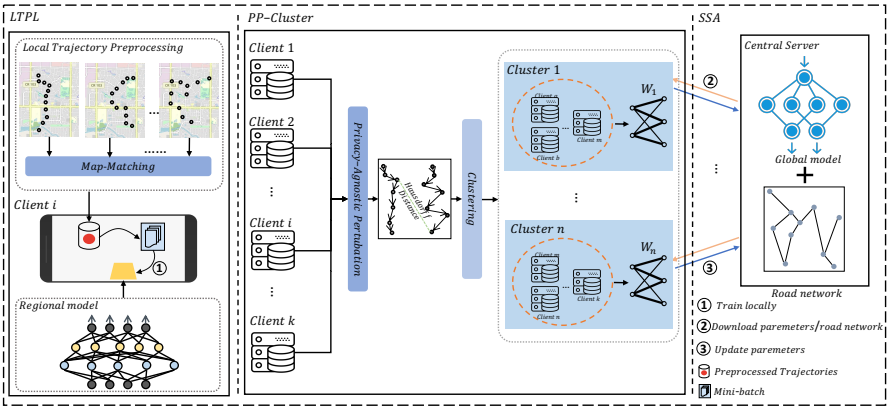


Fig. 4.2: F-STL Framework Overview

Definition 18 (Trajectory Similarity) *Given two trajectories T_i and T_j, the spatio-temporal similarity $\mathcal{D}(T_i, T_j)$ is defined as a weighted, linear combination of spatial and temporal similarity.*

$$\mathcal{D}(T_i, T_j) = \Lambda \cdot \mathcal{D}_{Spa}(T_i, T_j) + (1 - \Lambda) \cdot \mathcal{D}_{Tem}(T_i, T_j), \tag{4.3}$$

where $\mathcal{D}_{Spa}$ and $\mathcal{D}_{Tem}$ denote spatial and temporal similarity, respectively, and $\Lambda \in [0, 1]$ is a parameter controlling the contributions of spatial and temporal similarity, thus providing flexibility that can be used to support different applications as discussed in Section 1.

To define $\mathcal{D}$, we adopt an existing network-aware distance measure such as Two-Phase divide-and-conquer algorithm (TP) [56], Distributed In-Memory Trajectory Analytics (DITA) [57], Longest Common Road Segment (LCRS) [82]), and Network-aware Edit Distance with Real Penalty (NetERP) [33].

2.2 Problem Definition

Consider a user with personal trajectories as a client. Given a central server $\mathcal{S}$ with a road network G, and $\mathcal{K}$ clients with local trajectory datasets $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \cdots, \mathcal{T}_\mathcal{K}\}$, the problem is to learn a shared global score function $\mathcal{F}(\cdot, \cdot)$, such that for any pair of trajectories T_i and T_j from τ in each client, the difference $|\mathcal{F}(T_i, T_j) - \mathcal{D}(T_i, T_j)|$ is minimized. Note that each client keeps its local data and only transfers parameters to the central server during training by considering privacy.

3 Methodology

We detail the Federated Trajectory Similarity Learning (F-TSL) framework for decentralized trajectory similarity computation. F-TSL is based on horizontal learning and uses a client-server structure to collaboratively train local trajectory learning models under the orchestration of a central server, as illustrated in Figure 4.2. It consists of three major modules: local trajectory preprocessing and learning, privacy-preserving clustering, and server-side aggregation.

3.1 Local Trajectory Preprocessing and Learning

In this module, each client first preprocesses locally collected trajectory data. Meanwhile, it downloads parameters and the global road network and then maps all trajectories to the corresponding roads. Finally, the client learns a local model to quantify local trajectory similarity. Note that the model for the central server and the models for all clients share the same trajectory learning model structure, as shown in Figure 4.3. We elaborate the trajectory learning model in Section 3.1.

Local Trajectory Preprocessing

Given a road network that is downloaded from the central server, we first map all trajectories to the corresponding road intersections according to the Incremental Map-Matching (IMM) method [8]. This way, we convert the raw GPS data into time-ordered trajectories as follows.

$$T = \langle (p_1, t_1,), (p_2, t_2), \cdots, (p_n, t_n) \rangle, \tag{4.4}$$

where p_i is i-th point at time stamp t_i and n denotes the trajectory length. To tackle the issue of noisy data and to learn powerful trajectory representations, we remove trajectories with fewer than 10 points (i.e., $n \leq 10$). Further, to facilitate the capture of spatial and temporal correlations, we further separate T into $T_s = \langle p_1, p_2, \cdots, p_n \rangle$ and $T_t = \langle t_1, t_2, \cdots, t_n \rangle$ that represent the spatial and temporal aspects of T, respectively.

Local Trajectory Learning

Existing trajectory embedding methods focus predominantly on capturing spatial correlations of trajectories while ignoring temporal dependencies. To better capture spatial and temporal dependencies of trajectories, inspired by [24], we introduce a local trajectory learning method, including a spatio-temporal encoder and a spatio-temporal decoder, which uses each client's local data to learn effective trajectory embeddings.

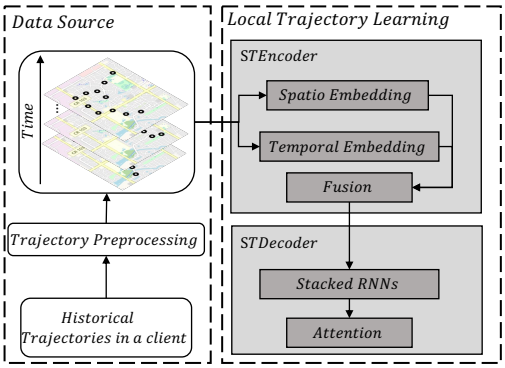


Fig. 4.3: Local Trajectory Learning Model

Spatio-Temporal (ST) Encoder. Given T_s and T_t, the ST encoder (see Figure 4.3) aims to learn spatial and temporal embeddings of trajectories. It first learns spatial and temporal embeddings and then combines them.

$$\begin{aligned} h_s &= SpatialEmbedding(T_s) \\ h_t &= TemporalEmbedding(T_t) \\ h &= h_s + h_t, \end{aligned} \tag{4.5}$$

where h_s and h_t denote the learned spatial and temporal embeddings, respectively, + is element-wise addition.

Spatial Embeddings. Spatial embeddings are designed to learn spatial correlations of trajectories. Given T_s, we first adopt Node2Vec [27] to capture the co-occurrence of the adjacent locations, which aims to learn road structure information. Specifically, given a point p_i, we obtain its embedding e_i by approximating the spatial conditional probability of its nearby points as follows.

$$e_i = Node2Vec(p_i) \tag{4.6}$$

After obtaining all the embeddings $emb = \langle e_1, e_2, \cdots, e_n \rangle$ of T_s, we input emb into stacked GCN layers to learn final spatial embeddings.

$$h_{p_i} = \sigma((e_i \oplus \sum_{j \in N_i} a_{i,j} W_s e_j) + b_{p_i}), \tag{4.7}$$

where $\oplus$ denotes the concatenation operation, N_i is the neighborhood set of p_i, $a_{i,j}$ is an adjacency weight, W_s is a learnable parameter, and b_{p_i} represents the bias. Based on Equations 4.6 and 4.7, we learn high-dimensional features, $h_s = \langle h_{p_1}, h_{p_2}, \cdots, h_{p_n} \rangle$, which captures spatial correlations of T_s.

Temporal Embeddings. Given T_t, we embed it to get a vector with size of $l+1$ for each timestamp t as follows.

$$s_t[i] = \begin{cases} w_i t + b_i, & i = 0 \\ cos(w_i t + b_i), & 1 \leq i \leq l, \end{cases} \tag{4.8}$$

where w_i and b_i represent learnable parameters and bias, respectively, and $cos(\cdot)$ is the cosine function. Based on Equation 4.8, we embed T_t into time vectors $h_t^s = \langle s_1, s_2, \cdots, s_n \rangle$. Then we input h_t^s into stacked recurrent neural networks (RNNs) to further learn high-dimensional temporal dependencies of T_t.

$$h_t = RNNs(h_t^s) \tag{4.9}$$

Spatio-Temporal (ST) Decoder. The learned spatio-temporal embeddings h (see Equation 4.5) are then input into the ST decoder to decode the data representations for similarity computation. As shown in Figure 4.3, the ST decoder contains several stacked RNNs, $RNNs(\cdot)$, followed by an attention layer, $Attention(\cdot)$, to learn a projection function for computing similarities. It can be formulated as follows.

$$\begin{aligned} h_{decode} &= RNNs(h) \\ v &= Attention(h_{decode}) = \sum_{i=0}^{n} \alpha_i h_{decode}[i], \end{aligned} \tag{4.10}$$

where $\alpha_i = \frac{exp(h_{decode}[i])}{\sum_{j=0}^{n} exp(h_{decode}[j])}$, and v is the learned embeddings.

Following previous studies [24, 29], the dot product is adopted to measure the similarity between embeddings of trajectories. Specifically, for a pair of trajectories T_i and T_j, we learn their respective embeddings v_{T_i} and v_{T_j}, and the similarity score $\mathcal{F}(T_i, T_j) = v_{T_i}^{\mathsf{T}} v_{T_j}$, where T is the transpose operation.

3.2 Privacy-Preserving Clustering

To alleviate the problem of data heterogeneity, we propose a novel privacy-preserving clustering mechanism to aggregate similar clients together by calculating the Hausdorff distance between locally perturbed embeddings of clients, aiming at achieving more focused model updates to ensure consistent model updates between the server and clients. Considering privacy protection, we first perturb the locally learned trajectory embeddings $V = \langle v_1, v_2, \cdots, v_l \rangle$ of each client to $V' = \langle v_1', v_2', \cdots, v_l' \rangle$ by means of differential privacy, where l is the length of the local dataset. Then, we partition all clients into clusters $\mathcal{M} = \{M_1, M_2, \cdots, M_{\mathcal{M}}\}$, where $\sum_{i=1}^{m} |M_i| = \mathcal{K}$, based on the Hausdorff distance between the perturbed embeddings of clients, which considers the information of individual trajectories.

We first give two necessary definitions: ϵ-differential privacy [21] and sensitivity.

Definition 1 (ϵ-differential privacy) *A random algorithm M is ϵ-differentially private if $\forall S \subseteq OutRange(M)$ and two datasets X and $Y = [X; z](z \in \mathbb{R}^d)$, we have*

$$Pr[M(x) \in S] \leq e^{\epsilon} Pr[M(Y) \in S]. \tag{4.11}$$

Definition 2 (sensitivity) *For a function f, the sensitivity of f is defined as follows.*

$$\Delta f = \max_{X,Y} ||f(X) - f(Y)||_1 \tag{4.12}$$

Privacy-Agnostic Perturbation. Using local data from clients directly when performing clustering may cause sensitive information leakage. To avoid this problem, we propose a privacy-agnostic perturbation method based on differential privacy (i.e., a Laplacian mechanism). In particular, given a client c_i with local trajectories $\mathcal{T}_i$ and local trajectory learning model $f(\cdot)$, we perturb the original trajectories by adding Laplacian noise to their learned high-dimensional embeddings V.

$$\begin{aligned} &M(\mathcal{T}_i) = f(\mathcal{T}_i) + z = V + z, \\ &z_i \sim Lap(0, \frac{\Delta f}{\epsilon}), z_i \in z \\ &V' = M(\mathcal{T}_i), \end{aligned} \tag{4.13}$$

where $z = (z_1, z_2, \cdots, z_d)$, d is the feature dimension of V, and $Lap(\cdot)$ is the Laplacian distribution.

Theorem 1 *$V' = M(\mathcal{T}_i)$ preserves $d\epsilon$-differential privacy.*

proof 1 *Let p_x and p_y denote the probability density of $M(x)$ and $M(y)$ at an arbitrary output v', respectively. We have:*

$$\begin{aligned} \frac{p_x(v')}{p_y(v')} &= \prod_{i=1}^{q} \left(\frac{\exp\left(-\frac{d\epsilon \left| f(x)_i - z_i \right|}{\Delta f}\right)}{\exp\left(-\frac{d\epsilon \left| f(y)_i - z_i \right|}{\Delta f}\right)} \right) \\ &= \prod_{i=1}^{q} \exp\left(\frac{d\epsilon \left(\left| f(y)_i - z_i \right| - \left| f(x)_i - z_i \right| \right)}{\Delta f} \right) \\ &\leq \prod_{i=1}^{q} \exp\left(\frac{d\epsilon \left| f(x)_i - f(y)_i \right|}{\Delta f} \right) \\ &= \exp\left(\frac{d\epsilon \cdot \| f(x) - f(y) \|_1}{\Delta f} \right) \leq \exp(d\epsilon), \end{aligned} \tag{4.14}$$

Algorithm 4 The Hausdorff Distance based Clustering

Input: a set of clients $\mathbf{C} = \{c_1, \cdots, c_{\mathcal{K}}\}$ with locally perturbed trajectory embeddings $V_i' \subseteq c_i$, the number of cluster centers $\mathcal{M}$
Output: $\mathcal{M}$ cluster centers: $M_1, M_2, \cdots, M_{\mathcal{M}}$
1: Randomly select a client c_i in $\mathbf{C}$ as the first cluster center M_1
2: $\overline{\mathbf{C}} \leftarrow C - c_i$
3: **for** c_j in $\overline{C}$ **do**
4: Compute Hausdorff distance between embeddings of client $c_j (V_j' \subseteq c_j)$ and cluster $M_1 (V_i' \subseteq c_i)$ by Equation 4.15
5: $Dis(c_j) \leftarrow HD(V_j', V_i')$
6: **end for**
7: Sort $\overline{\mathbf{C}}$ by $Dis(\cdot)$ in descending order
8: Select top $\mathcal{M} - 1$ clients $\mathbf{C}_{ini}$ in sorted $\overline{\mathbf{C}}$ as remaining initial cluster centers
9: $\overline{\mathbf{C}} \leftarrow \overline{\mathbf{C}} - \mathbf{C}_{ini}$
10: **while** $\mathcal{M}$ is changed **do**
11: **for** c_k $(V_k' \subseteq c_k)$ in $\overline{\mathbf{C}}$ **do**
12: **for** M_m $(V_m' \subseteq M_m)$ in $\mathcal{M}$ **do**
13: Compute Hausdorff distance between embeddings of client c_k and cluster center $M_{\mathcal{M}}$ by Equation 4.15
14: $Dis_c(M_m) \leftarrow HD(V_k', V_m')$
15: **end for**
16: $m^* \leftarrow argmin_m Dis_c(M_m)$ % *Assign c_k to closest center*
17: $M_{m^*} \leftarrow M_{m^*} \cup c_k$
18: $V_{m^*}' \leftarrow V_{m^*}' \cup V_k'$
19: **end for**
20: **end while**

where x and y are two trajectories in $\mathcal{T}_i$, and q is the number of the feature dimensions.

We use the Laplacian mechanism to perturb embeddings for all clients. Specifically, for client c_i, we perturb embeddings $V' = \langle v_1', v_2', \cdots, v_l' \rangle$ for the following Hausdorff distance based clustering.

Hausdorff Distance based Clustering. Most existing federated clustering methods [5, 18] consider a client as a whole and perform clustering at the client level, ignoring the influence of fine-grained local data (i.e., client-specific trajectories), which hurts trajectory learning performance [14]. To solve this problem, we design a Hausdorff distance based clustering mechanism to aggregate similar clients in high-dimensional feature spaces, by measuring the maximum mismatch degree among trajectory embedding sets in clients.

The Hausdorff distance based clustering is shown in Algorithm 4. Specifically, we first select $\mathcal{M}$ cluster centers that are far away from each other in the high-dimensional space (lines 1–9). Then, for unselected client c_j, we compute the Hausdorff distance among embeddings of client c_j and $\mathcal{M}$ cluster centers in the high-dimensional feature space (lines 12–15). The computation of Hausdorff distance is given by Equation 4.15. Finally, we group the client with its

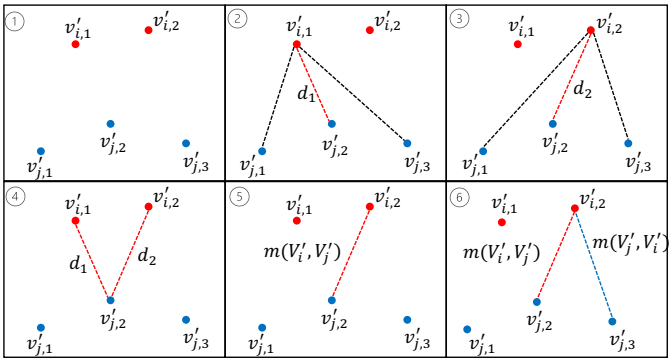


Fig. 4.4: Hausdorff Distance Computation

nearest cluster center (lines 16–18) until cluster centers are unchanged.

Hausdorff Distance Computation. Given two clients with local perturbed embeddings, V'_i and V'_j, we compute their Hausdorff distance $HD(\cdot)$ as follows.

$$\begin{aligned} HD(V'_i, V'_j) &= \max(m(V'_i, V'_j), m(V'_j, V'_i)), \\ m(V'_i, V'_j) &= \max_{v'_i \in V'_i} (\min_{v'_j \in V'_j} (dist(v'_i, v'_j))), \\ m(V'_j, V'_i) &= \max_{v'_j \in V'_j} (\min_{v'_i \in V'_i} (dist(v'_j, v'_i))), \end{aligned} \tag{4.15}$$

where $dist(\cdot)$ is the Euclidean distance between high-dimensional perturbed trajectory embeddings.

Figure 4.4 shows an example of Hausdorff distance computation between two local embedding sets $V'_i = \langle v'_{i,1}, v'_{i,2} \rangle$ and $V'_j = \langle v'_{j,1}, v'_{j,2}, v'_{j,3} \rangle$. For embedding $v'_{i,1}$ in V'_i, we calculate the Euclidean distance between $v'_{i,1}$ and any embedding in V'_j iteratively, where d_1 is the minimum distance. Similarly, we find that d_2 is the minimum distance between $v_{i,2}$ and V'_j, where $d_2 > d_1$. Then, we use d_2 for $m(V'_i, V'_j)$, which is the shortest distance between V'_i and V'_j. Since we have a directed road network, we obtain the shortest distance $m(V'_j, V'_i)$ between V'_j and V'_i similarly. Finally, it is easy to see that $m(V'_i, V'_j)$ is smaller than $m(V'_j, V'_i)$. Thus we use $m(V'_j, V'_i)$ as the Hausdorff distance between V'_i and V'_j.

3.3 Server-Side Aggregation

We propose a server-side aggregation method to collaboratively train the clustered clients, which first aggregates parameter updates on a local cluster with a subset of clients and then performs parameter aggregation among clusters.

Algorithm 5 Hierarchical Parameter Update

Input: R: communication rounds, θ_s^0: initial global model of server, M: cluster numbers, E: local training epochs, α: learning rate
Output: θ_s: final global model of server

for each round r in $[1, 2, \cdots, R]$ **do**
  Server transmits θ_s^{r-1} to M cluster centers
  $\Theta_m \leftarrow \theta_s^0$ for cluster M_m in M clusters
  for M_m in M clusters **do**
    Sample F fraction of clients in M_m
    Transmit Θ_m to selected clients in M_m
    for each selected client $c_i \in M_m$ **do**
      for each local epoch e in $[1, \cdots, E]$ **do**
        $\theta_{c_i}^r \leftarrow arg\min_\theta L_{local}(\theta_{c_i}^r)$
        $\nabla L_{local}(\theta_{c_i}^r) \leftarrow \nabla L_{local}(\theta_{c_i}^{r-1}) - \alpha(\theta_{c_i}^r - \theta_{c_i}^{r-1})$
      end for
      $\theta_{c_i} \leftarrow \theta_{c_i}^r$
      Transmit local client model θ_{c_i} to the respective cluster
    end for
    $\Theta_m \leftarrow \sum_{i=1}^{|M_m|} \frac{|c_i|}{|M_m|} \theta_{c_i}$
  end for
  $\theta_s \leftarrow \sum_{m=1}^{M} \frac{1}{M} \Theta_m$
end for

In federated trajectory similarity learning, we have $\mathcal{K}$ clients, denoted as $\{c_1, c_2, \cdots, c_\mathcal{K}\}$, each with its own dataset $\mathcal{T}_i$ that has three parts: a training, a validation, and a test dataset, denoted as $\mathcal{T}_i^{train}$, $\mathcal{T}_i^{valid}$, and $\mathcal{T}_i^{test}$, respectively. We aim to learn a good trajectory recovery model $\mathcal{F}_i(\cdot)$ for each client c_i on its local dataset $\mathcal{T}_i$:

$$min \frac{1}{N} \sum_{i=1}^{N} \frac{1}{n_i^{test}} \sum_{j=1}^{n_i^{test}} L_{local}(\mathcal{F}_i(T_j, T_n, T_q)), \tag{4.16}$$

where L_{local} is the local loss function, to be detailed in Equation 4.17, n_i^{test} is the length of $\mathcal{T}_i^{test}$, and T_n and T_p are sampled similar and dissimilar trajectories of T_i, respectively.

Next, we introduce the local objective function of each client and hierarchical parameters updating in a federated training way.

Local Objective Function. The local objective function contains two parts, a loss of trajectory similarity L_{sim} and a client-cluster alignment loss L_{align}, for alleviating data heterogeneity. We combine them together, and the overall local loss of a client is as follows.

$$L_{local} = L_{sim} + L_{align} \tag{4.17}$$

Following existing studies [24, 29] on trajectory similarity, we adopt a

triplet-based pair-wise loss for effective and robust training.

$$L_{sim} = \max \sum_{T_m \in \mathcal{T}} \sum_{T_n \in \mathcal{T}, T_q \in \mathcal{T} \setminus \{T_m\}} \mathbb{I}(\mathcal{F}(T_m, Tn) < \mathcal{F}(T_m, T_q)), \tag{4.18}$$

where T_m is a randomly selected trajectory in $\mathcal{T}$, T_n and T_p are sampled similar and dissimilar trajectories of T_m, respectively, and $\mathbb{I}(\cdot)$ is an indicator function, where it equals one if the condition (i.e., $\mathcal{F}(T_m, Tn) < \mathcal{F}(T_m, T_q)$) holds, otherwise it is zero. Note that we randomly sample the trajectory of T_m instead of traversing all trajectories to improve efficiency.

To contend with data heterogeneity, inspired by the Moreau envelopes in personalized federated learning [61], we propose a simple but effective client-cluster alignment loss to balance the updates between clients and the cluster center. For a client c_i in cluster M_j, we have

$$L_{align} = \frac{\lambda}{2} ||\theta_i - \Theta_j||^2, \tag{4.19}$$

where λ is a control weight, θ_i is the local model of client c_i, and Θ_j is an aggregated model of cluster j. We combine L_{sim} and L_{align} to train a suitable local model for a client and ensure it not drift off the aggregated model of its corresponding cluster.

Hierarchical Parameter Updating. We propose a hierarchical parameter updating method to perform clustered federated training, as shown in Algorithm 5. In particular, at each communication round r, the central server transmits its current model θ_s^{r-1} to each cluster (lines 1–3). Each cluster transmits its current parameters Θ_m to a subset (F fraction) of the clients, aiming to reduce the training time (lines 4–6). Each active client optimizes a local empirical risk objective, which is the sum of similarity loss and alignment loss (lines 7–11). Next, the cluster updates its universal model (lines 12–15) to proceed with more focused model updates, i.e., the model updates of clients are consistent with those of the server. Finally, the central server aggregates parameters from clusters to update its model θ_s until it converges to a stationary point θ_s (line 17).

4 Experimental Evaluation

4.1 Experimental Setup

Datasets

The experiments are carried out on two widely-used public trajectory datasets: Tdrive[1] and TaxiRome[2].

Baselines

We compare the proposed F-TSL with the following baseline methods.

- **FC+FL**. The FC+FL method employs stacked fully connected (FC) networks as the local model trained with FedAvg [42] under the horizontal federated learning (FL) framework.
- **RNN+FL**. The RNN+FL combines horizontal FL with a stacked RNN to train a decentralized model for trajectory similarity computation.
- **GCN+FL**. GCN+FL is a decentralized trajectory similarity computation model, which uses stacked graph convolutional network (GCN) integrated with horizontal FL to collaboratively learn trajectory embeddings.

Evaluation Metrics

Following existing learning-based trajectory similarity studies [24, 29], *top-k* similarity search is utilized for experimental validation. We use *HR*@10, *HR*@50, and *R10*@50 as evaluation metrics. Specifically, *HR@k* denotes the *top-k* hitting ratio that is the overlap degree between a *top-k* result and the corresponding ground truth. For each client, we have

$$HR@K = \frac{1}{\mathcal{T}^{te}} \sum_{T \in \mathcal{T}^{te}} \frac{|L_T^P@K \cap L_T^G|}{|L_T^G|}, \tag{4.20}$$

where $\mathcal{T}^{te}$ is the local test set of trajectories, $L_T^P@K$ is the list of the predicted trajectories that are most similar to a given trajectory T with length K, L_T^G is the set of similar trajectories in the testing set for the given trajectory T, and $|\cdot|$ is set cardinality. *Rk@t* is the *top-t* recall for the *top-k* ground truth that captures the fraction of the *top-k* ground truth in the corresponding top-t result. In addition, we adopt four traditional trajectory similarity measure methods to obtain the ground truth results of top-k similarity search, i.e., Two-Phase

[1] https://www.microsoft.com/en-us/research/publication/t-drive-trajectory-data-sample/
[2] https://crawdad.org/roma/taxi/20140717/

Table 4.1: Overall Accuracy on Two Datasets

Datasets	Baselines	TP			DITA			LCRS			NetERP		
		HR@10	HR@50	R10@50	HR@10	HR@50	R10@50	HR@10	HR@50	R10@50	HR@10	HR@50	R10@50
Tdrive	FC+FL	0.0087	0.0230	0.0279	0.0165	0.0351	0.0556	0.0059	0.0208	0.0267	0.0412	0.0739	0.1018
	RNN+FL	0.1120	0.1753	0.2928	**0.1476**	0.1507	0.1566	0.1028	0.1983	0.2214	0.1061	0.1162	**0.2077**
	GCN+FL	0.0078	0.0224	0.0320	0.0071	0.0246	0.0312	0.0106	0.0241	0.0354	0.0251	0.0621	0.0713
	F-TSL	**0.1769**	**0.2585**	**0.3709**	0.1366	**0.1670**	**0.2520**	**0.1330**	**0.2088**	**0.3531**	**0.1496**	**0.1745**	0.1816
TaxiRome	FC+FL	0.0204	0.0524	0.0752	0.0246	0.0593	0.0833	0.0165	0.0384	0.0567	0.0589	0.0110	0.1659
	RNN+FL	0.1059	0.1894	0.2690	0.0619	0.1241	0.1663	0.0582	0.0964	0.1683	0.0514	0.0889	0.1311
	GCN+FL	0.0068	0.0231	0.0278	0.0095	0.0324	0.0358	0.0068	0.0268	0.0314	0.0279	0.0803	0.0929
	F-TSL	**0.1663**	**0.2472**	**0.3396**	**0.1278**	**0.1881**	**0.2633**	**0.0698**	**0.1033**	**0.1749**	**0.0819**	**0.1256**	**0.1765**

divide-and-conquer algorithm (**TP**) [56], Distributed In-Memory Trajectory Analytics (**DITA**) [57], Longest Common Road Segment (**LCRS**) [82]) and Network-aware Edit Distance with Real Penalty (**NetERP**) [33]. The larger the $HR@10$, the $HR@50$, and the $R10@50$ are, the more accurate the method is.

4.2 Experimental Results

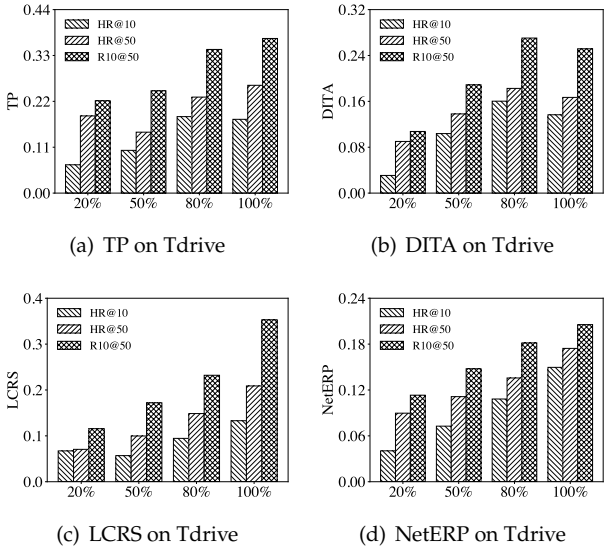
Overall Accuracy

Table 4.1 shows the performance comparison among different methods on the two datasets. The best performance by an existing method (FC+FL, RNN+FL, and GCN+FL) is underlined, and the overall best performance is marked in bold.The observations are as follows.

- Our F-TSL achieves the best results among all the baselines in most cases on the two datasets. F-TSL performs better than the best among the baselines by up to 57.94% (HR@10), 60.92% (R10@50), 59.48% (R10@50), and 50.17% (HR@50) for TP, DITA, LCRS, and NetERP, respectively on Tdrive. We observe that the performance improvements obtained by F-TSL on Tdrive exceed those on TaxiRome, due to the fact that Tdrive has much more training data than TaxiRome. Thus, the methods trained with sufficient data get better results.

- RNN+FL, which can capture temporal dependencies of trajectories, performs much better than FC+FL and GCN+FL (that can capture spatial dependencies) on both Tdrive and TaxiRome. It shows that effectively capturing temporal dependencies of trajectories is more important than learning spatial correlations since the originally collected trajectories are preprocessed into sequence data.

- Compared with the state-of-the-art centralized method ST2Vec, F-TSL performs worse than ST2Vec over Tdrive, while performs comparably over TaxiRome since it is easier for the centralized method to learn commonalities from centralized data with shuffled mini-batch training. But the centralized methods hurt privacy severely, which makes the learned model vulnerable.



(a) TP on Tdrive (b) DITA on Tdrive

(c) LCRS on Tdrive (d) NetERP on Tdrive

Fig. 4.5: Effect of Client Fractions on Tdrive

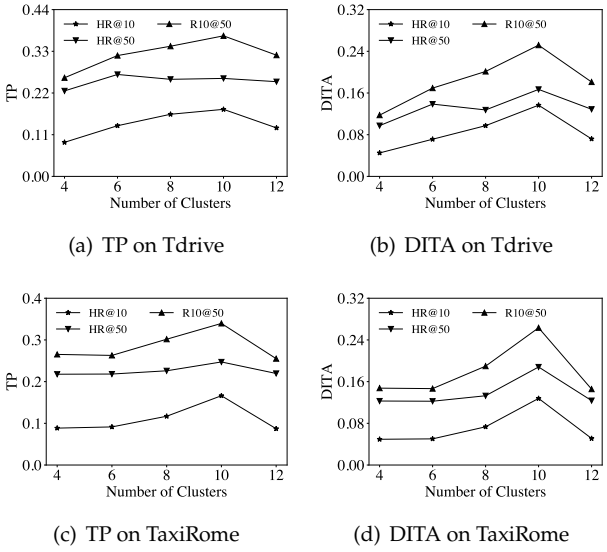
The above observations indicate that F-TSL is more effective than the existing FL-based methods.

Effect of Client Fractions

When aggregating parameters in F-TSL, we sample a certain fraction of clients to perform model training in each cluster. To study the effect of client fraction F on the model performance, we conduct experiments by sampling four different fractions of clients: 20%, 50%, 80%, and 100%. The results are shown in Figure 4.5. Generally, the model performance gets better with the increase of client fractions. The performance is improved by at least 39.65% evaluated by four different metrics (i.e., TP, DITA, LCRS, and NetERP) when the client fraction changes from 20% to 100%. It shows that a bigger fraction of clients with more training data can lead to better model performance since more useful knowledge can be learned.

Effect of the Number of Clusters

We next study how sensitive the F-TSL is w.r.t. the number of clusters, where we consider 100 clients and 4, 6, 8, 10, and 12 clusters. The results are given in Figure 4.6. We observe that all curves first increase slightly and then drop. It is reasonable to use 10 clusters in this set of experiments. When the number of clusters changes from 10 to 12, the model performance decreases as it is more difficult to make consistent updates between the server and clients. This leads



(a) TP on Tdrive (b) DITA on Tdrive

(c) TP on TaxiRome (d) DITA on TaxiRome

Fig. 4.6: Effect of the Number of Clusters on Both Datasets (100 Clients)

to ineffective feature learning. To save space, we only report the results of TP and DITA on the two datasets; similar results are obtained in the evaluation of LCRS and NetERP. Also, similar results are obtained when setting the number of clients to 50 and 80.

Centralized Method vs. F-TSL

To compare the proposed F-TSL with the centralized method in terms of trajectory similarity learning, we select ST2Vec [24], a recent state-of-the-art deep learning framework for centralized trajectory similarity computation, which considers fine-grained spatial and temporal relations between trajectories. To save space, we only report the results of LCRS and NetERP, in Table 4.2. We can see that F-TSL generally performs worse than ST2Vec on Tdrive and TaxiRome since it is easier for the centralized method to learn commonalities from centralized data with shuffled mini-batch training. But the centralized method reduces privacy severely, which makes the learned model vulnerable. In addition, we observe that F-TSL performs comparably with ST2Vec on TaxiRome. This is because that data contained in TaxiRome is sparse, and the proposed privacy-preserving clustering mechanism in F-TSL can alleviate the influence of sparsity and achieve more effective feature learning.

Table 4.2: Centralized Method vs. F-TSL

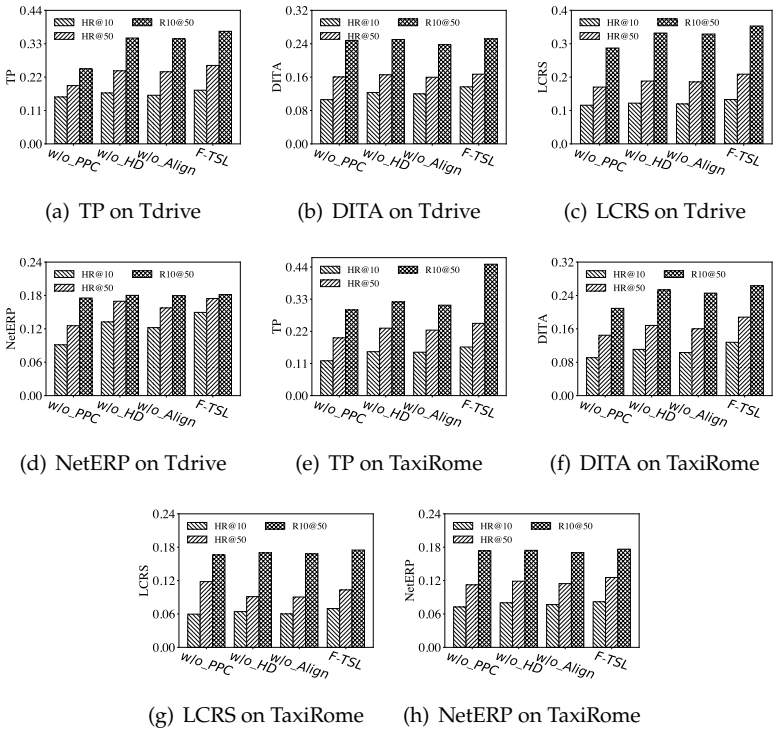
Datasets	Metrics		Method	
			ST2Vec	F-TSL
Tdrive	LCRS	HR@10	0.1973	0.1330
		HR@50	0.1734	0.2088
		R10@50	0.2704	0.3531
	NetERP	HR@10	0.1495	0.2433
		HR@50	0.1744	0.3170
		R10@50	0.1816	0.4166
TaxiRome	LCRS	HR@10	0.0621	0.0698
		HR@50	0.1013	0.1033
		R10@50	0.1629	0.1749
	NetERP	HR@10	0.0892	0.0819
		HR@50	0.1217	0.1256
		R10@50	0.1909	0.1765

Ablation Study

To gain insight into the effects of key aspects of F-TSL, we evaluate three F-TSL variants.

- ***w/o Privacy-Preserving Clustering (w/o_PPC)***. *w/o_PPC* is the F-TSL model without the privacy-preserving clustering mechanism.
- ***w/o Hausdorff Distance (w/o_HD)***. *w/o_HD* replaces the Hausdorff distance with l_1*-norm* distance to compute the similarity between trajectory embeddings.
- ***w/o Alignment Loss (w/o_Align)***. *w/o_Align* is the F-TSL model without the alignment loss L_{align}.

To evaluate whether the components in F-TSL all contribute to the performance of trajectory similarity learning, we compare F-TSL with its variants. The results are shown in Figure 4.7. Regardless of the datasets, F-TSL always outperforms its counterparts without privacy-preserving clustering, Hausdorff distance, or alignment loss, which shows that these three components are all useful to trajectory similarity learning since removing any one of them will decrease the prediction accuracy. Further, on both datasets, *w/o_PPC* performs the worst among all variants in most cases. F-TSL outperforms *w/o_PPC*, improving the performance by up to 49.79%, 28.30%, 22.78%, and 63.74% in terms of TP, DITA, LCRS, and NetERP, respectively, which offers evidence of the effectiveness of the proposed privacy-preserving clustering mechanism.



(a) TP on Tdrive (b) DITA on Tdrive (c) LCRS on Tdrive

(d) NetERP on Tdrive (e) TP on TaxiRome (f) DITA on TaxiRome

(g) LCRS on TaxiRome (h) NetERP on TaxiRome

Fig. 4.7: Performance of F-TSL and Its Variants on Both Datasets

Chapter 5

Conclusion and Future Work

1 Conclusion

This thesis explores deep spatio-temporal analytics for time series, streaming, and trajectory data, presenting four research studies that introduce novel, advanced spatio-temporal networks. The key contributions of each study are summarized as follows.

- **Paper A** [45] introduces TimeDC, an efficient time series dataset condensation framework that endeavors to synthesize a small yet informative dataset summarizing the original, large time series dataset. To capture complex temporal dependencies, we design a time series feature extraction module with stacked *TSOperators*. In addition, we propose decomposition-driven frequency matching to ensure similar temporal patterns between the condensed and original datasets. To enable effective and generalized dataset condensation, we feature a curriculum training trajectory matching module with an expert buffer that aims to reduce training costs.
- **Paper B** [48] proposes a unified replay-based continuous learning framework for spatio-temporal prediction on streaming data, designed to capture complex patterns in spatio-temporal data. To embrace historical knowledge, an STMixup mechanism is designed to integrate current training data with representative samples from a replay buffer. By coupling a spatio-temporal SimSiam network with a prediction autoencoder, holistic features are extracted through mutual information maximization. In addition, five spatio-temporal data augmentation methods are proposed to enhance the performance of the SimSiam network.
- **Paper C** [46] introduces LightTR+, a novel federated framework for

lightweight incremental trajectory recovery that aims to enable decentralized trajectory learning while ensuring privacy protection. To enhance computational efficiency, we propose a lightweight local trajectory embedding module with a lightweight *ST-operator* for each client, which effectively extracts features. We propose an intra-domain knowledge distillation module to alleviate intra-domain interference. In addition, a novel meta-knowledge enhanced local-global training module is proposed to improve communication efficiency between the server and clients, thereby reducing data heterogeneity across clients and mitigating inter-domain interference.

- **Paper D** [44] proposes a federated trajectory similarity learning framework, entitled F-TSL, for decentralized trajectory similarity computation, which is based on horizontal federated learning to ensure privacy protection. To capture complex spatio-temporal correlations in trajectories, each client is equipped with a local trajectory preprocessing and learning module featuring a spatio-temporal encoder and decoder. A novel privacy-preserving clustering mechanism is proposed to content with trajectory heterogeneity. In addition, an alignment loss is designed to maintain consistent model updates between the server and clients.

2 Future Work

As for future work, several promising directions could be considered. First, most existing spatio-temporal prediction methods for streaming data are replay-based, which can be computationally expensive. Enhancing training efficiency by advanced methods (e.g., model pruning and curriculum learning) in streaming spatio-temporal prediction would be valuable for real-world applications. Second, spatio-temporal data is ubiquitous across various domains such as retail, finance, manufacturing, and healthcare. Developing a foundation model for streaming spatio-temporal data that can identify the commonalities across diverse domains would be an interesting area for further research.

References

[1] C. C. Aggarwal and P. S. Yu, "On static and dynamic methods for condensation-based privacy-preserving data mining," *TODS*, vol. 33, no. 1, pp. 1–39, 2008.

[2] J. A. Alvarez-Garcia, J. A. Ortega, L. Gonzalez-Abril, and F. Velasco, "Trip destination prediction based on past GPS log using a hidden markov model," *Expert Systems with Applications*, vol. 37, no. 12, pp. 8166–8171, 2010.

[3] S. Atev, G. Miller, and N. P. Papanikolopoulos, "Clustering of vehicle trajectories," *TITS*, vol. 11, no. 3, pp. 647–657, 2010.

[4] L. Bai, L. Yao, C. Li, X. Wang, and C. Wang, "Adaptive graph convolutional recurrent network for traffic forecasting," in *NeurIPS*, 2020, pp. 17 804–17 815.

[5] M.-F. Balcan, T. Dick, Y. Liang, W. Mou, and H. Zhang, "Differentially private clustering in high-dimensional euclidean spaces," in *ICML*, 2017, pp. 322–331.

[6] P. Banerjee, S. Ranu, and S. Raghavan, "Inferring uncertain trajectories from partial observations," in *ICDM*, 2014, pp. 30–39.

[7] A. Bonifati, F. D. Buono, F. Guerra, and D. Tiano, "Time2feat: learning interpretable representations for multivariate time series clustering," *PVLDB*, vol. 16, no. 2, pp. 193–201, 2022.

[8] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk, "On map-matching vehicle tracking data," in *PVLDB*, 2005, pp. 853–864.

[9] G. Cazenavette, T. Wang, A. Torralba, A. A. Efros, and J.-Y. Zhu, "Dataset distillation by matching training trajectories," in *CVPR*, 2022, pp. 4750–4759.

[10] E. Chambers, B. T. Fasy, Y. Wang, and C. Wenk, "Map-matching using shortest paths," *TSAS*, vol. 6, no. 1, pp. 1–17, 2020.

[11] Y. Chang, J. Qi, Y. Liang, and E. Tanin, "Contrastive trajectory similarity learning with dual-feature attention," in *ICDE*, 2023, pp. 2933–2945.

[12] O. Chapelle, J. Weston, L. Bottou, and V. Vapnik, "Vicinal risk minimization," in *NeurIPS*, 2000, pp. 1–7.

[13] F. Chen, M. Luo, Z. Dong, Z. Li, and X. He, "Federated meta-learning with fast convergence and efficient communication," *arXiv preprint arXiv:1802.07876*, 2018.

[14] J. Chen, R. Wang, L. Liu, and J. Song, "Clustering of trajectories based on hausdorff distance," in *ICECC*, 2011, pp. 1940–1944.

[15] M. Chen, Y. Zhao, Y. Liu, X. Yu, and K. Zheng, "Modeling spatial trajectories with attribute representation learning," *TKDE*, vol. 34, no. 4, pp. 1902–1914, 2022.

[16] X. Chen and K. He, "Exploring simple siamese representation learning," in *CVF*, 2021, pp. 15 750–15 758.

[17] Y. Chen, H. Zhang, W. Sun, and B. Zheng, "RNTrajRec: Road network enhanced trajectory recovery with spatial-temporal transformer," *arXiv:2211.13234*, 2022.

[18] V. Cohen-Addad, A. Epasto, S. Lattanzi, V. Mirrokni, A. Munoz Medina, D. Saulpic, C. Schwiegelshohn, and S. Vassilvitskii, "Scalable differentially private clustering via hierarchically separated trees," in *SIGKDD*, 2022, pp. 221–230.

[19] M. Delange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. Slabaugh, and T. Tuytelaars, "A continual learning survey: Defying forgetting in classification tasks," *TPAMI*, vol. 44, no. 7, pp. 3366–3385, 2022.

[20] L. Deng, H. Sun, Y. Zhao, S. Liu, and K. Zheng, "S2tul: A semi-supervised framework for trajectory-user linking," in *WSDM*, 2023, pp. 375–383.

[21] C. Dwork, "Differential privacy," in *ICALP*, 2006, pp. 1–12.

[22] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *TCC*, 2006, pp. 265–284.

[23] Z. Fang, Q. Long, G. Song, and K. Xie, "Spatial-temporal graph ode networks for traffic flow forecasting," in *SIGKDD*, 2021, pp. 364–373.

[24] Z. Fang, Y. Du, X. Zhu, D. Hu, L. Chen, Y. Gao, and C. S. Jensen, "Spatio-temporal trajectory similarity learning in road networks," in *SIGKDD*, 2022, pp. 347–356.

[25] Z. Fang, S. Gong, L. Chen, J. Xu, Y. Gao, and C. S. Jensen, "Ghost: A general framework for high-performance online similarity queries over distributed trajectory streams," *SIGMOD*, vol. 1, no. 2, pp. 1–25, 2023.

[26] R. Z. Farahani and M. Hekmatfar, *Facility location: concepts, models, algorithms and case studies*. Springer Science & Business Media, 2009.

[27] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *SIGKDD*, 2016, pp. 855–864.

[28] Y. Guo, B. Liu, and D. Zhao, "Online continual learning through mutual information maximization," in *ICML*, 2022, pp. 8109–8126.

[29] P. Han, J. Wang, D. Yao, S. Shang, and X. Zhang, "A graph-based approach for trajectory similarity computation in spatial networks," in *SIGKDD*, 2021, pp. 556–564.

[30] S. Hoteit, S. Secci, S. Sobolevsky, C. Ratti, and G. Pujolle, "Estimating human trajectories and hotspots through mobile phone data," *Comput. Netw.*, vol. 64, pp. 296–307, 2014.

[31] S. K. Jensen, T. B. Pedersen, and C. Thomsen, "Modelardb: Modular model-based time series management with spark and cassandra," *PVLDB*, vol. 11, no. 11, pp. 1688–1701, 2018.

[32] K. Kalpakis, D. Gada, and V. Puttagunta, "Distance measures for effective clustering of arima time-series," in *ICDM*, 2001, pp. 273–280.

[33] S. Koide, C. Xiao, and Y. Ishikawa, "Fast subtrajectory similarity search in road networks under weighted edit distance constraints," *PVLDB*, vol. 13, no. 11, pp. 2188–2201, 2020.

[34] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," in *ICLR*, 2018, pp. 1–10.

[35] Y. Li, Y. Shen, and L. Chen, "Camel: Managing data for efficient stream learning," in *SIGMOD*, 2022, pp. 1271–1285.

[36] Y. LIANG, "Learning human mobility with deep spatio-temporal neural networks," 2022.

[37] M. Lippi, M. Bertini, and P. Frasconi, "Short-term traffic flow forecasting: An experimental comparison of time-series analysis and supervised learning," *TITS*, vol. 14, no. 2, pp. 871–882, 2013.

[38] S. Liu, H. Su, Y. Zhao, K. Zeng, and K. Zheng, "Lane change scheduling for autonomous vehicle: A prediction-and-search framework," in *SIGKDD*, 2021, pp. 3343–3353.

[39] Z. Liu, H. Miao, Y. Zhao, C. Liu, K. Zheng, and H. Li, "LightTR: A lightweight framework for federated trajectory recovery," in *ICDE*, 2024.

[40] D. Madaan, J. Yoon, Y. Li, Y. Liu, and S. J. Hwang, "Representational continuity for unsupervised continual learning," in *ICLR*, 2021, pp. 1–18.

[41] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *AISTATS*, 2017, pp. 1273–1282.

[42] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *AISTATS*, 2017, pp. 1273–1282.

[43] C. Meng, S. Rambhatla, and Y. Liu, "Cross-node federated graph neural network for spatio-temporal data modeling," in *SIGKDD*, 2021, pp. 1202–1211.

[44] H. Miao, Z. Liu, Y. Zhao, , K. Zheng, Y. Zhang, and C. S. Jensen, "Federated trajectory similarity learning with privacy-preserving clustering," *in submission*, 2024.

[45] H. Miao, Z. Liu, Y. Zhao, C. Guo, B. Yang, K. Zheng, and C. S. Jensen, "Less is more: Efficient time series dataset condensation via two-fold modal matching," *PVLDB*, 2025.

[46] H. Miao, Z. Liu, Y. Zhao, C. Liu, C. Guo, B. Yang, K. Zheng, H. Li, and C. S. Jensen, "LightTR+: A lightweight incremental framework for federated trajectory recovery," *in submission*, 2024.

[47] H. Miao, J. Shen, J. Cao, J. Xia, and S. Wang, "MBA-STNet: Bayes-enhanced discriminative multi-task learning for flow prediction," *TKDE*, vol. 14, no. 8, pp. 1–14, 2022.

[48] H. Miao, Y. Zhao, C. Guo, B. Yang, K. Zheng, F. Huang, J. Xie, and C. S. Jensen, "A unified replay-based continuous learning framework for spatio-temporal prediction on streaming data," in *ICDE*, 2024, pp. 1050–1062.

[49] K. L. Musliner, T. Munk-Olsen, W. W. Eaton, and P. P. Zandi, "Heterogeneity in long-term trajectories of depressive symptoms: Patterns, predictors and outcomes," *Journal of affective disorders*, vol. 192, pp. 199–211, 2016.

[50] OpenAI, "Chatgpt (oct 10 version)," 2024, large language model. [Online]. Available: https://chat.openai.com/

[51] S. J. Pan and Q. Yang, "A survey on transfer learning," *TKDE*, vol. 22, no. 10, pp. 1345–1359, 2009.

[52] E. Rappos, S. Robert, and P. Cudré-Mauroux, "A force-directed approach for offline GPS trajectory map matching," in *SIGSPATIAL*, 2018, pp. 319–328.

[53] H. Ren, S. Ruan, Y. Li, J. Bao, C. Meng, R. Li, and Y. Zheng, "MTrajRec: Map-constrained trajectory recovery via seq2seq multi-task learning," in *SIGKDD*, 2021, pp. 1410–1419.

[54] L. Rui, X. Huang, S. Song, Y. Kang, C. Wang, and J. Wang, "Time series representation for visualization in apache iotdb," *SIGMOD*, vol. 2, no. 1, pp. 1–26, 2024.

[55] J. Serra, D. Suris, M. Miron, and A. Karatzoglou, "Overcoming catastrophic forgetting with hard attention to the task," in *ICML*, 2018, pp. 4548–4557.

[56] S. Shang, L. Chen, Z. Wei, C. S. Jensen, K. Zheng, and P. Kalnis, "Trajectory similarity join in spatial networks," *PVLDB*, vol. 10, no. 11, pp. 1178–1189, 2017.

[57] Z. Shang, G. Li, and Z. Bao, "Dita: distributed in-memory trajectory analytics," in *SIGMOD*, 2018, pp. 725–740.

[58] S. Shekhar and B. M. Williams, "Adaptive seasonal time series models for forecasting short-term traffic flow," *TRR*, vol. 2024, no. 1, pp. 116–125, 2007.

[59] B. L. Smith and M. J. Demetsky, "Traffic flow forecasting: comparison of modeling approaches," *Journal of transportation engineering*, vol. 123, no. 4, pp. 261–266, 1997.

[60] H. Sun, C. Yang, L. Deng, F. Zhou, F. Huang, and K. Zheng, "Periodicmove: Shift-aware human mobility recovery with graph neural network," in *CIKM'21*, 2021, pp. 1734–1743.

[61] C. T Dinh, N. Tran, and J. Nguyen, "Personalized federated learning with moreau envelopes," *NeurIPS*, vol. 33, pp. 21 394–21 405, 2020.

[62] K. S. Tai, V. Sharan, P. Bailis, and G. Valiant, "Sketching linear classifiers over data streams," in *SIGMOD*, 2018, pp. 757–772.

[63] J. Taylor, X. Zhou, N. M. Rouphail, and R. J. Porter, "Method for investigating intradriver heterogeneity using vehicle trajectory data: A dynamic time warping approach," *Transportation Research Part B: Methodological*, vol. 73, pp. 59–80, 2015.

[64] Y. Tong, D. Shi, Y. Xu, W. Lv, Z. Qin, and X. Tang, "Combinatorial optimization meets reinforcement learning: Effective taxi order dispatching at large-scale," *TKDE*, pp. 1–1, 2021.

[65] S. Tuli, G. Casale, and N. R. Jennings, "Tranad: deep transformer networks for anomaly detection in multivariate time series data," *PVLDB*, vol. 15, no. 6, pp. 1201–1214, 2022.

[66] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *NeurIPS*, vol. 30, 2017.

[67] C. Wang, X. Huang, J. Qiao, T. Jiang, L. Rui, J. Zhang, R. Kang, J. Feinauer, K. A. McGrail, P. Wang *et al.*, "Apache iotdb: Time-series database for internet of things," *PVLDB*, vol. 13, no. 12, pp. 2901–2904, 2020.

[68] D. Wang, L. Wu, D. Zhang, J. Zhou, L. Sun, and Y. Fu, "Human-instructed deep hierarchical generative learning for automated urban planning," in *AAAI*, vol. 37, no. 4, 2023, pp. 4660–4667.

[69] Q. Wang, S. Whitmarsh, V. Navarro, and T. Palpanas, "iedeal: A deep learning framework for detecting highly imbalanced interictal epileptiform discharges," *PVLDB*, vol. 16, no. 3, pp. 480–490, 2022.

[70] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. S. Quek, and H. Vincent Poor, "Federated learning with differential privacy: Algorithms and performance analysis," *IEEE T INF FOREN SEC*, vol. 15, pp. 3454–3469, 2020.

[71] M. Welling, "Herding dynamical weights to learn," in *ICML*, 2009, pp. 1121–1128.

[72] H. Wu, J. Xu, J. Wang, and M. Long, "Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting," *NeurIPS*, vol. 34, pp. 22 419–22 430, 2021.

[73] X. Wu, D. Zhang, M. Zhang, C. Guo, B. Yang, and C. S. Jensen, "AutoCTS+: Joint neural architecture and hyperparameter search for correlated time series forecasting," *SIGMOD*, vol. 1, no. 1, pp. 1–26, 2023.

[74] Z. Wu, S. Pan, G. Long, J. Jiang, X. Chang, and C. Zhang, "Connecting the dots: Multivariate time series forecasting with graph neural networks," in *SIGKDD*, 2020, pp. 753–763.

[75] Z. Wu, S. Pan, G. Long, J. Jiang, and C. Zhang, "Graph wavenet for deep spatial-temporal graph modeling," in *IJCAI*, 2019, pp. 1907–1913.

[76] T. Xia, Y. Qi, J. Feng, F. Xu, F. Sun, D. Guo, and Y. Li, "AttnMove: History enhanced trajectory recovery via attentional network," in *AAAI*, vol. 35, no. 5, 2021, pp. 4494–4502.

[77] J. Xiao, Y. Huang, C. Hu, S. Song, X. Huang, and J. Wang, "Time series data encoding for efficient storage: A comparative analysis in apache iotdb," *PVLDB*, vol. 15, no. 10, pp. 2148–2160, 2022.

[78] B.-K. Yi, H. V. Jagadish, and C. Faloutsos, "Efficient retrieval of similar time sequences under time warping," in *ICDE*, 1998, pp. 201–208.

[79] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, "Graph contrastive learning with augmentations," in *NeurIPS*, 2020, pp. 5812–5823.

[80] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting," in *IJCAI*, 2018, pp. 3634–3640.

[81] X. Yu, Y. Peng, F. Li, S. Wang, X. Shen, H. Mai, and Y. Xie, "Two-level data compression using machine learning in time series database," in *ICDE*, 2020, pp. 1333–1344.

[82] H. Yuan and G. Li, "Distributed in-memory trajectory similarity search and join on road network," in *ICDE*, 2019, pp. 1262–1273.

[83] J. Yuan, Y. Zheng, C. Zhang, X. Xie, and G.-Z. Sun, "An interactive-voting based map matching algorithm," in *MDM*, 2010, pp. 43–52.

[84] J. Zhang, Y. Zheng, and D. Qi, "Deep spatio-temporal residual networks for city-wide crowd flows prediction," in *AAAI*, 2017, pp. 1655–1661.

[85] L. Zhang, Z. Deng, K. Kawaguchi, A. Ghorbani, and J. Zou, "How does mixup help with robustness and generalization?" *ICLR*, 2020.

[86] B. Zhao, K. R. Mopuri, and H. Bilen, "Dataset condensation with gradient matching," in *ICLR*, 2021. [Online]. Available: https://openreview.net/forum?id=mSAKhLYLSsl

[87] X. Zhao, W. Fan, H. Liu, and J. Tang, "Multi-type urban crime prediction," in *AAAI*, vol. 36, no. 4, 2022, pp. 4388–4396.

[88] K. Zheng, Y. Zheng, X. Xie, and X. Zhou, "Reducing uncertainty of low-sampling-rate trajectories," in *ICDE*, 2012, pp. 1144–1155.

[89] K. Zheng, G. Chen, M. Herschel, K. Y. Ngiam, B. C. Ooi, and J. Gao, "Pace: learning effective task decomposition for human-in-the-loop healthcare delivery," in *SIGMOD*, 2021, pp. 2156–2168.

[90] S. Zheng, Y. Yue, and P. Lucey, "Generating long-term trajectories using deep hierarchical networks," in *NIPS*, 2016, p. 1551–1559.

[91] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, "Informer: Beyond efficient transformer for long sequence time-series forecasting," in *AAAI*, vol. 35, no. 12, 2021, pp. 11 106–11 115.

[92] Y. Zhou, E. Nezhadarya, and J. Ba, "Dataset distillation using neural feature regression," *NeurIPS*, vol. 35, pp. 9813–9827, 2022.

Part II

Papers

Paper A

Less is More: Efficient Time Series Dataset Condensation via Two-fold Modal Matching

Hao Miao, Ziqiao Liu, Yan Zhao, Chenjuan Guo, Bin Yang, Kai Zheng, Christian S. Jensen

The paper has been accepted by the
51st International Conference on Very Large Data Bases, 2025.

© 201X IEEE
The layout has been revised.

Abstract

The expanding instrumentation of processes throughout society with sensors yields a proliferation of time series data that may in turn enable important applications, e.g., related to transportation infrastructures or power grids. Machine-learning based methods are increasingly being used to extract value from such data. We provide means of reducing the resulting considerable computational and data storage costs. We achieve this by providing means of condensing large time series datasets such that models trained on the condensed data achieve performance comparable to those trained on the original, large data. Specifically, we propose a time series dataset condensation framework, TimeDC, that employs two-fold modal matching, encompassing frequency matching and training trajectory matching. Thus, TimeDC performs time series feature extraction and decomposition-driven frequency matching to preserve complex temporal dependencies in the reduced time series. Further, TimeDC employs curriculum training trajectory matching to ensure effective and generalized time series dataset condensation. To avoid memory overflow and to reduce the cost of dataset condensation, the framework includes an expert buffer storing pre-computed expert trajectories. Extensive experiments on real data offer insight into the effectiveness and efficiency of the proposed solutions.

1 Introduction

With the proliferation of edge computing and mobile sensing, massive volumes of time series data, comprising millions of observations, are being collected and stored into time series database systems (TSMSs) [1, 2], enabling various real-world applications [3–5]. We are seeing impressive advances in machine learning [6, 7], especially in deep learning [3, 8], that are successful at extracting information and creating value from large time series datasets. However, the proposed methods are also resource-intensive. Thus, storing and preprocessing datasets is costly, and model training often calls for the use of specialized equipment and infrastructure [9–11], limiting the application on edge devices [2, 12].

An effective way to reduce costs associated with the use of large data is coreset construction [9, 13–15], which often involves clustering [16], Gaussian mixture models [17], and streaming learning [9, 18]. Coreset construction methods [16, 17] often define heuristic criteria to select the most representative subsets from full datasets to form small coresets, such that models trained on the coresets are competitive with those built on the full datasets. Unfortunately, heuristic coreset construction methods guarantee neither optimal solutions nor promise the presence of representative observations in relation to downstream tasks [19]. Given these limitations, a recent alternative approach, dataset condensation [10, 20] aims to directly synthesize small condensed, optimized

datasets, not relying on representative subset-selection.

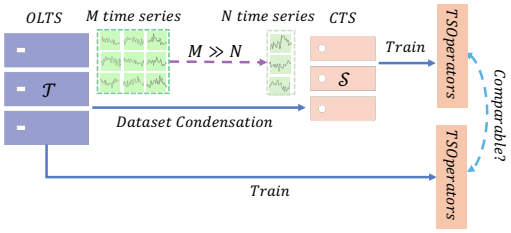


Fig. A.1: Time Series Condensation

We consider the novel problem of time series dataset condensation, where the goal is to synthesize a small but informative condensed time series dataset $\mathcal{S}$ derived from an original large dataset $\mathcal{T}$. A model, composed of stacked time series operators (*TSOperators*) that perform feature extraction, is trained on the condensed time series (CTS) dataset with the objective of achieving performance comparable to that of a model trained on the original large time series (OLTS) dataset for downstream tasks (see Figure A.1). The stacked *TSOperators* are multiple *TSOperators* arranged sequentially so that the output of one *TSOperator* is fed into the next *TSOperator*. Stacked *TSOperators* enable more complex feature extraction. Time series are usually stored in TSMSs over edge devices [1, 12]. In the setting of continuously produced time series, time series dataset condensation aims to alleviate the storage burden of TSMSs considering the limited processing capability of edge devices, which further contributes to the *database community* [21, 22]. Although substantial research has been devoted to inventing effective dataset condensation methods [10, 19, 20, 23, 24], such as gradient matching [10], distribution matching [20], and multi-step parameters matching [24], existing methods target image data and cannot be applied to time series data directly as they are not built to contend with the unique temporal dependencies present in time series [8, 25]. These methods thus fail to capture inherent temporal dependencies such as trends and seasonalities [26, 27]. New methods are needed to enable time series condensation. However, developing such methods is non-trivial, due to the following challenges.

Challenge I: Effectiveness and Generalization Ability. It is challenging to guarantee the effectiveness and generalization ability of the condensed time series dataset [21, 24]. First, typical dataset condensation methods require a bi-level (even triple-level) optimization to jointly learn a minimum of two objectives: model parameters and a condensed time series dataset. Such complex non-convex optimization cannot guarantee optimal solutions, thus significantly limiting its effectiveness as the representative of the large original time series dataset. Second, the condensed time series datasets should be generalized to train different networks. Nonetheless, the condensed data may suffer from

various types of overfitting [24], e.g., overfitting to a certain network architecture. Also, the downstream models might overfit the condensed data during training.

Challenge II: Complex Temporal Dependencies. Existing popular dataset condensation methods mainly focus on computer vision without specific modules to capture the unique time series characteristics [3, 28]. Existing dataset condensation methods are ill-equipped for modeling the complex temporal dependencies of time series. For example, traffic conditions during afternoon rush hour may be similar on consecutive workdays. Moreover, morning rush hours may gradually start later when winter arrives as people get up later and later due to the gradual decrease in temperature and later sunrise. It is important to learn complex temporal dependencies in a time series dataset so that a condensed version of the dataset exhibits similar temporal patterns to the original dataset. In addition, time series datasets are often multivariate, encompassing correlated variables, or features (channels). Such features are often coupled to perform feature extraction simultaneously. We argue that analysis in independent channels enables more effective feature extraction. For example, traffic flow is negatively correlated with vehicle speed, but positively correlated with road occupancy [29]. Coupling features with varying and complex interactions might confuse a model, leading to decreased model performance.

Challenge III: Scalability. Existing dataset condensation methods often suffer from poor scalability, as bi-level optimization is generally time-consuming [30]. For example, gradient-matching methods [10] set different hyper-parameters of the outer and the inner loop optimization for different learning settings, which requires extensive cross-validation, rendering condensed dataset synthesis costly. In addition, these methods may incur memory overflow: they require an entire model to reside in memory during training, which is infeasible in many resource-constrained environments [31]. It is highly desirable, but also non-trivial, to develop an efficient time series dataset condensation method.

This study addresses the above challenges by providing an efficient time series dataset condensation (TimeDC) framework, which features a two-fold matching mechanism: time series frequency matching and training trajectory matching. TimeDC synthesizes a small time series dataset that summarizes a large dataset such that models trained on the small dataset achieve comparable performance to models trained on the large dataset. It encompasses three major modules: a time series feature extraction module, a decomposition-driven frequency matching module, and a curriculum training trajectory matching module. Time series of stock prices are among the most common time series in real-world applications. We thus use such time series in a running example to to explain and convey the intuition underlying definitions and the proposed modules throughout the paper.

To achieve effective and generalized TS dataset condensation (*Challenge I*), we propose a curriculum training trajectory matching (CT^2M) module (see Figure A.4), where we consider an original time series dataset as the gold standard and seek to imitate the long-term training dynamics of models trained on it. The parameter trajectories trained on the original dataset are called expert trajectories. CT^2M first trains a set of expert trajectories of a model, composed of a set of stacked *TSOperators*, on the large dataset. These are computed offline and are stored in an expert buffer and then serve as guidance for the condensed dataset optimization. Next, we sample trajectories from the expert buffer with curriculum trajectory queries and conduct long-term trajectory matching to align the segments of trajectories between the condensed time series dataset and the large dataset, enabling comprehensive and general knowledge transfer to the condensed dataset.

To support the capture of complex temporal dependencies (*Challenge II*), we propose a time series feature extraction (TSFE) module. In particular, TSFE contains a channel-independent mechanism and a set of stacked *TSOperators*, where a *TSOperator* includes a set of self-attention and fully connected layers. A decomposition block separates and refines the frequency, i.e., trend and seasonality, progressively from the intermediate results extracted by each *TSOperator*. We design a decomposition-driven frequency matching (DDFM) module to facilitate consistent temporal pattern preservation between the condensed time series dataset and the original one. It maps the decomposed frequencies when optimizing the condensed dataset.

To address the issue of high computation costs and poor scalability (*Challenge III*), we propose an expert buffer to save expert trajectories pre-computed on original time series data to avoid memory overflow and reduce the overall training time. Further, we employ a patching mechanism that splits the time series into patches, thereby accelerating time series feature extraction in the TSFE module and also enabling effective local semantics modeling for time series.

The main contributions are summarized as follows.

- To the best of our knowledge, this is the first study to learn dataset condensation over time series. We propose a framework called TimeDC that aims to condense large time series datasets into small synthetic time series datasets while retaining the expressiveness of the large datasets.
- We design a TSFE module to capture temporal dependencies of time series effectively. We further propose a DDFM module to reduce the discrepancy of frequencies between original and condensed time series.
- A novel curriculum trajectory query and matching module is proposed to penalize condensed time series data based on how far the synthetically trained model deviates from expert trajectories.

- We report on experiments using real data, offering evidence of the effectiveness and efficiency of the proposals.

The remainder of this paper is organized as follows. Section 2 covers preliminary concepts and formalizes the problem of time series dataset condensation. We detail the TimeDC framework in Section 3, followed by the experimental study in Section 4. Section 5 surveys related work, and Section 6 concludes the paper.

2 Preliminaries

We proceed to present the necessary preliminaries and then define the problem addressed.

Definition 19 (Time Series) *A time series $T = \langle t_1, t_2, \cdots, t_n \rangle$ is a time ordered sequence of n observations, where each observation $t_i \in \mathbb{R}^C$ is a C-dimensional vector. If $C = 1$, T is univariate; otherwise, T is multivariate. For example, a series of stock prices with multiple features, e.g., the opening price, closing price, highest price, and lowest price, is an example of multivariate time series*

Definition 20 (Time Series Dataset) *A time series dataset $\mathcal{T}$ is a set of time series $\mathcal{T} = \{T_1, T_2, \cdots, T_M\}$, where M is the cardinality.*

2.1 Time Series Seasonal-Trend Decomposition via Moving Average

Time series exhibit distinct frequencies [26, 32], which are typically decomposed into seasonalities and trends in an additive form.

$$T_i = TRE_i + SEA_i + R_i, \tag{A.1}$$

where T_i is the time series in time interval i ($0 < i < n$) with a sliding window w ($0 < w < n$), TRE_i is a trend-cycle component encompassing the trend and cycle (we use *trend-cycle* and *trend* interchangeably when the meaning is clear from the context), SEA_i is a seasonal component, and R_i denotes the remainder that consists mostly of noise. The intuition behind the seasonal-trend decomposition is that *trend captures the continuous increase or decrease*, while *seasonality captures periodic patterns*.

In real-world applications as well as our study, in order to obtain robust and optimal performance, application of denoising techniques in a preprocessing step is indispensable [26]. Also, we observe empirically that the decomposed noise is small and can be ignored. The trend and seasonality are sufficient

to capture the primary patterns of the preprocessed time series. Thus, Equation A.1 can be simplified as follows [8, 33],

$$T_i \approx TRE_i + SEA_i,\ i = 1, 2, \cdots, n, \tag{A.2}$$

where $0 < i < n$.

Moving average is an effective means of obtaining trend information from time series [33, 34]. Given a time series with n observations and a sliding window of size w, the moving average (i.e., trend) over the i-th sliding window is obtained as follows.

$$TRE_i = \frac{t_{i-w+1} + t_{i-w+2} + \cdots + t_i}{w} = \frac{1}{w} \sum_{j=i-w+1}^{i} t_j, \tag{A.3}$$

where $0 < i < n$ and $0 < w < n$. Following Equation A.2, the seasonality is obtained as $SEA_i = T_i - TRE_i$.

2.2 Dataset Condensation over Time Series

Definition 21 (Condensed Time Series Dataset) *Given a time series dataset $\mathcal{T}$, a time series dataset $\mathcal{S} = \left\{\widetilde{T}_1, \widetilde{T}_2, \cdots, \widetilde{T}_N\right\}$ is condensed if N is much smaller than M, where $\widetilde{T}$ is the condensed time series, M is the cardinality, and $\mathcal{S}$ is derived from $\mathcal{T}$.*

Note that we can initialize each item in $\mathcal{S}$ with a random time series in the original dataset or a Gaussian noise. After dataset condensation, we expect to learn an optimized $\mathcal{S}$ to replace $\mathcal{T}$.

Given a large time series dataset $\mathcal{T} = \{T_1, T_2, \cdots, T_M\}$, we aim to learn an optimal differentiable function f parameterized by θ that correctly performs time series tasks, e.g., time series forecasting. We use $\mathcal{T}$ as a training set to learn the optimal parameters $\theta^{\mathcal{T}}$ by minimizing an empirical loss term as follows.

$$\theta^{\mathcal{T}} = \arg\min_{\theta} \mathcal{L}^{\mathcal{T}}(\theta), \tag{A.4}$$

where $\mathcal{L}^{\mathcal{T}}(\theta) = \frac{1}{M} \sum_{T_i \in \mathcal{T}} \ell(f_\theta, T_i)$, and $\ell(\cdot)$ is a task specific loss (e.g., Mean Squared Error). We denote the generalization performance of the obtained model $f_{\theta^{\mathcal{T}}}$ by $\mathbb{E}_{T_i \sim P_{\mathcal{T}}}[\ell(f_{\theta^{\mathcal{T}}}, T_i)]$, where $P_{\mathcal{T}}$ is the data distribution of $\mathcal{T}$.

Dataset Condensation over Time Series. Given a large time series dataset $\mathcal{T}$, our goal is to synthesize a small time series dataset $\mathcal{S} = \left\{\widetilde{T}_1, \widetilde{T}_2, \cdots, \widetilde{T}_N\right\}$ that preserves most of the information in $\mathcal{T}$ and $N \ll M$. Once the condensed time series set $\mathcal{S}$ is learned, we can train f on it by replacing $\mathcal{T}$ by $\mathcal{S}$ in Equation A.4. As the condensed set $\mathcal{S}$ is significantly smaller than $\mathcal{T}$, we expect the optimization to be considerably faster. In addition, we expect the

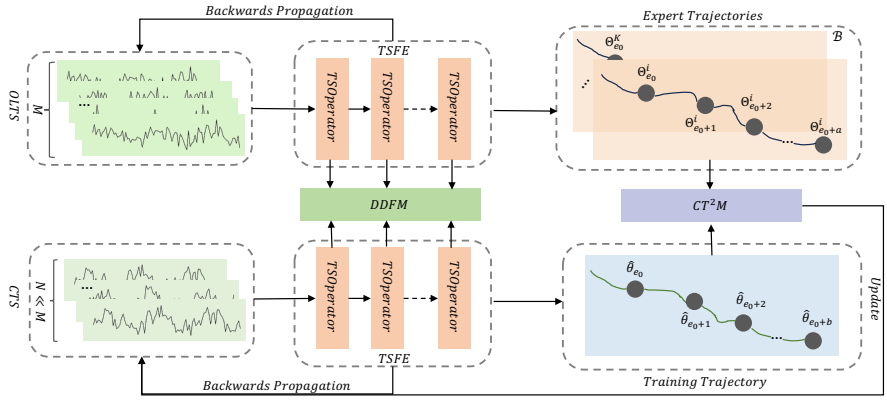


Fig. A.2: TimeDC Framework Overview

generalization performance of f_{θ^S} to be close to $f_{\theta^\mathcal{T}}$, i.e., $\mathbb{E}_{T_i \sim P_\mathcal{T}}[\ell(f_{\theta^\mathcal{T}}, T_i)] \simeq \mathbb{E}_{T_i \sim P_\mathcal{S}}[\ell(f_{\theta^S}, T_i)]$, on the real data distribution $P_\mathcal{T}$.

Thus, given a learnable function f parameterized by θ, our problem, *dataset condensation over time series*, can be formulated as a bi-level optimization problem [10] as follows.

$$\overbrace{\min_{\mathcal{S}} \mathcal{L}(f_{\theta^S}, \mathcal{T})}^{outer-level}, \; s.t. \; \theta^\mathcal{S} = \overbrace{\arg\min_{\theta} \mathcal{L}(f_\theta, \mathcal{S})}^{inner-level}, \tag{A.5}$$

where the outer loop optimizes the condensed time series set $\mathcal{S}$, while the inner loop aims to learn an optimal function f_{θ^S} on $\mathcal{S}$.

Offline Expert Trajectories. We denote the time sequence of parameters $\{\theta_e^k\}_{e=1}^E$ as expert trajectories, that are obtained during the training of the proposed TSFE module on the full, original time series dataset, where E is the training epoch intervals. To generate these expert trajectories, we train TSFE several times on original datasets and save their snapshot parameters at every epoch. We call these parameter sequences as expert trajectories since they represent the theoretical upper bound for the downstream applications.

3 Methodology

We proceed to detail the efficient time series dataset condensation framework, TimeDC. We first give an overview of the framework and then provide specifics on each module in the framework.

3.1 Framework Overview

As illustrated in Figure A.2, given a model composed of stacked *TSOperators*, we first pre-train a set of training trajectories on the original large time series (OLTS) dataset and store these pre-trained trajectories as expert trajectories in an expert buffer. As the original dataset is used to guide the network training,

we denote the parameters trained on the original dataset as expert trajectory. Then, we train the same model on the condensed time series (CTS) dataset and perform two-fold modal matching: trajectory matching and frequency matching. Specifically, we optimize the condensed dataset with respect to the distance between the synthetically training trajectories and the trajectories trained on the original dataset. Further, we align the frequencies (i.e., trend and seasonality) between the original dataset and the condensed dataset to preserve temporal correlations when synthesizing the condensed dataset. The framework encompasses three major modules: time series feature extraction (TSFE), decomposition-driven frequency matching (DDFM), and curriculum training trajectory matching (CT^2M).

- *Time Series Feature Extraction.* This module aims to extract effective high-dimensional features from the input time series. A channel-independent mechanism is adopted to decouple the time series $T_{input} \in \mathbb{R}^{\mathcal{B}\times n\times C}(C \geq 1)$ into C univariate time series $\{T^c_{input}\}^C_{c=1} \in \mathbb{R}^{\mathcal{B}\times n\times 1}$ for subsequent feature extraction. Next, the module splits each decoupled univariate (i.e., channel-independent) time series into P patches to learn local semantic information employing the patching mechanism. Then, the module stacks several *TSOperators*, composed of self-attention and fully connected layers, for channel-independent feature extraction. Finally, the channel-independent features are concatenated to enable prediction.

- *Decomposition-Driven Frequency Matching.* This module adopts a decomposition-driven frequency matching mechanism to reduce the discrepancy of decomposed frequencies between the original time series dataset and the condensed time series dataset, aiming to preserve temporal dependencies from the original dataset when synthesizing the condensed dataset. Specifically, it incorporates a moving-average based [26, 32] series decomposition block into each *TSOperator*, which separates the trend and seasonal information progressively from the learned features. This decomposition block enables a *TSOperator* to alternately decompose and refine the intermediate results during the time series feature extraction.

- *Curriculum Training Trajectory Matching.* This module first trains a set of training trajectories (i.e., parameters) of stacked *TSOperators* on the large original time series dataset, which are stored in an expert buffer. In addition, we design a curriculum trajectory query and matching mechanism to organize training trajectories matching from similar (easy) to dissimilar (hard). In particular, we sample trajectories from the expert buffer with minimally modified parameters to those with highly modified parameters, iteratively.

  Next, we provide the technical details of each module.

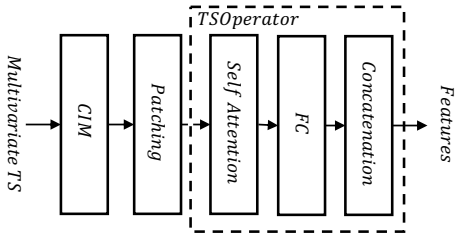


Fig. A.3: Feature Extraction Module

3.2 Time Series Feature Extraction

We first consider the time series feature extraction (TSFE) module. As shown in Figure A.3, TSFE is composed of a channel-independent mechanism, a patching mechanism, and stacked *TSOperators*.

Channel Independent Mechanism. Most existing time series modeling methods [3, 33] perform time series feature extraction by coupling all features and projecting them together into an embedding space. However, simply coupling different features may influence model performance very uncontrollably [29]. To avoid this problem, we employ the channel-independent mechanism $CIM(\cdot)$ to model each time series feature independently.

We consider a time series with C features as a sequence of multi-channel variables. Specifically, given a time series $T_{input} = \langle t_1, t_2, \cdots, t_n \rangle \in \mathbb{R}^{\mathcal{B} \times n \times C} (C \geq 1)$, the channel independent mechanism separates T_{input} into C univariate time series along the feature (i.e., channel) dimension, where $\mathcal{B}$ is the batch size. For example, in a series of stock prices with multiple features, e.g., opening prices and closing prices, the channel-independent mechanism separates these features by forming univariate time series. This means that the opening and closing prices are considered as two different in the subsequent feature extraction. Formally, the channel-independent mechanism works as follows.

$$CIM(T_{input}) = T^1_{input}, \cdots, T^c_{input}, \cdots, T^C_{input}, \tag{A.6}$$

where $T^c_{input} \in \mathbb{R}^{\mathcal{B} \times n \times 1}$. The separated univariate time series are then fed independently into the stacked *TSOperators*. The *TSOperators* share the same architecture, but their forward processes are independent.

Patching Mechanism. Time series feature extraction aims to model the correlations between observations at different time steps. We aggregate observations over several time steps of the separated univariate time series T^c_{input} into subseries-level patches [35] to enable local semantics modeling, which allows the model to see the longer historical sequences to improve feature extraction. For instance, in the context of stock prices, if the original time series represents hourly prices over a week, the patching mechanism may create patches where each patch contains the stock prices for each day, enhancing the capture of local semantics capturing.

We feed the univariate time series obtained from the channel independent mechanism into the patching mechanism that then divides them into patches that can be set to be overlapping or non-overlapping. Given a patch length L and a stride (i.e., the non-overlapping region between two consecutive patches) S, a sequence of patches $T_p^c \in \mathbb{R}^{\mathcal{B}\times L\times \mathcal{P}}$ is generated, where $\mathcal{P} = \lfloor \frac{n-L}{S} + 2 \rfloor$ is the number of patches. The last value t_n repeated $S-1$ times is padded to the end of the input time series before patching.

The patching decreases the memory usage and computational complexity of the following *TSOperator*, containing self-attention and fully connected (FC) layers quadratically, by a factor of S since the input time series length reduced from n to approximately n/S.

TSOperator. We then feed the patches T_p^c into the stacked *TSOperators* to learn latent features for the following frequency matching and trajectory matching. As illustrated in Figure A.3, a *TSOperator* includes a multi-head self-attention layer followed by a fully-connected (FC) layer. In the context of stock prices, the multi-head self-attention mechanism learns the temporal correlations between different time steps. This may include how the opening prices at the beginning of a day influence the prices in the next few days. The FC layer summarizes these temporal correlations for future prediction. The *j-th TSOperator* can be formulated as follows.

$$h^j = TSOperator(h^{j-1}) = Norm(FC(MultiHead(h^{j-1}))), \tag{A.7}$$

where $h^0 = T_p^c$, and $Norm(\cdot)$ is the normalization layer (i.e., BatchNorm). The multi-head self-attention layer has three components: *Query*, *Key*, and *Value*. The multi-head self-attention is computed by concatenating the output matrix of each attention head. For *i-th* attention head, the input latent features h^j are transformed as follows.

$$\begin{aligned} A_i^j &= Attention(h^j W_i^Q, h^j W_i^K, h^j W_i^V) \\ Attention(Q, K, V) &= softmax(\frac{Q \cdot K^T}{\sqrt{d_k}}) \cdot V \\ h^{j+1} &= Norm(h^j + FC_i(A_i^j h^j)), \end{aligned} \tag{A.8}$$

where d_k is a scaling factor, and W_i^Q, W_i^K, and W_i^V are the linear transformation parameter of the *Query*, *Key*, and *Value*, respectively.

The time series feature extraction is shown in Algorithm 6. Specifically, we first separate the batch of input time series T_{input} into C univariate time series (line 1) and apply the patching mechanism to split the univariate time series into patches (line 2). Then, we input the patches obtained in line 2 into the stacked *TSOperators* for feature extraction (lines 3–10). Finally, C extracted features are concatenated to get the overall hidden features h (line 12). The space and time complexities of the TSFE module are both $O(\frac{n^2}{S^2})$.

Algorithm 6 Time Series Feature Extraction

Input: a batch of time series: $T_{input} \in \mathcal{B} \times n \times C$; the number of *TSOperators*: N_{op}
Output: extracted features: h.

$T^1_{input}, \cdots, T^c_{input}, \cdots, T^C_{input} \leftarrow$ Separate T_{input} into C univariate channel-independent time series with Equation A.6
Split each univariate time series into patches
for $1 \leq c \leq C$ **do**
    $h^0_c \leftarrow T^c_{input}$;
    for $0 < j < N_{op}$ **do**
        Feature extraction with Equation A.7;
        $h^j_c \leftarrow TSOperator(h^{j-1}_c)$;
    end for
    $h_c \leftarrow h^{N_{op}-1}_c$;
end for
$h \leftarrow$ Concatenate $\{h_c\}^C_{c=1}$ along the channel;
return h

3.3 Decomposition-Driven Frequency Matching

Time series data usually exhibit distinct frequencies, such as trend and seasonality, separating from other modalities (e.g., images). For instance, the trend in stock prices indicates the prolonged direction of prices over time, which can be upward or downward. Seasonality refers to the regular, periodic fluctuations in stock prices, often influenced by seasonal business cycles or holidays. We argue that it would give more guidance on time series condensation if the condensed and original time series share similar frequencies. Thus, to preserve temporal patterns in the condensed time series dataset, we propose a novel decomposition-driven frequency matching (DDFM) mechanism that incorporates a decomposition block into the stacked *TSOperators*. The space and time complexities of DDFM are $O(n)$ and $O(n \cdot w)$, respectively.

As shown in Figure A.2, for each *TSOperator*, DDFM decomposes the intermediate hidden features, learned from the original dataset $\mathcal{T}$ and the condensed dataset $\mathcal{S}$, into their frequencies (i.e., trend and seasonality) progressively. It then aligns the frequencies to reduce the discrepancy of the temporal patterns between the original dataset and the condensed dataset based on the moving average. Concretely, for the hidden features h^j of the *j-th TSOperator*, we adopt the moving average by means of average pooling $AvgPool(\cdot)$ to smooth periodic fluctuations and highlight the trend and seasonality. In particular, the process of series decomposition is as follows.

$$h^j_{TRE} = AvgPool(Padding(h^j)),\ h^j_{SEA} = h^j - h^j_{TRE}, \tag{A.9}$$

where h^j_{TRE} and h^j_{SEA} are the decomposed trend and seasonality, respec-

tively. We employ the average pooling $AvgPool(\cdot)$ with the padding operation $Padding(\cdot)$ to keep the time series length unchanged. We denote the decomposition process as follows.

$$h^j_{TRE}, h^j_{SEA} = SeriesDecomposition(h^j) \tag{A.10}$$

Given the extracted features $h^j_{\mathcal{T}}$ and $h^j_{\mathcal{S}}$ of the original dataset $\mathcal{T}$ and the condensed dataset $\mathcal{S}$, we decompose the corresponding trend and seasonality according to Equation A.10, We use cosine similarity $cos(\cdot,\cdot)$ to measure similarities between the trends $h^j_{TRE_{\mathcal{T}}}$, $h^j_{TRE_{\mathcal{S}}}$ and seasonalities $h^j_{SEA_{\mathcal{T}}}$, $h^j_{SEA_{\mathcal{S}}}$, formulated as follows.

$$\begin{aligned} cos(h^j_{TRE_{\mathcal{T}}}, h^j_{TRE_{\mathcal{S}}}) &= \frac{h^j_{TRE_{\mathcal{T}}}}{||h^j_{TRE_{\mathcal{T}}}||_2} \cdot \frac{h^j_{TRE_{\mathcal{S}}}}{||h^j_{TRE_{\mathcal{S}}}||_2} \\ cos(h^j_{SEA_{\mathcal{T}}}, h^j_{SEA_{\mathcal{S}}}) &= \frac{h^j_{SEA_{\mathcal{T}}}}{||h^j_{SEA_{\mathcal{T}}}||_2} \cdot \frac{h^j_{SEA_{\mathcal{S}}}}{||h^j_{SEA_{\mathcal{S}}}||_2}, \end{aligned} \tag{A.11}$$

where $||\cdot||_2$ is l_2-*norm*.

We aim to maximize the cosine similarity to reduce the discrepancy of frequencies between the condensed data and the original data. The objective function of DDFM is formulated as follows.

$$L_{Fre} = -\frac{1}{N_{op}} \sum_{j}^{N_{op}} (cos(h^j_{TRE_{\mathcal{T}}}, h^j_{TRE_{\mathcal{S}}}) + cos(h^j_{SEA_{\mathcal{T}}}, h^j_{SEA_{\mathcal{S}}})), \tag{A.12}$$

where N_{op} is the number of *TSOperators*.

3.4 Curriculum Training Trajectory Matching

To enable effective dataset condensation, we propose a curriculum training trajectory matching mechanism. Existing popular dataset condensation methods, which are often based on gradient matching [10], conduct online gradient matching step by step. These methods have drawbacks when used for short-range matching, causing short-sightedness issues, thus failing to imitate holistic learning behaviors, and reducing the quality of condensed time series data.

Motivated by an existing study [24], we match the long-term training trajectories of the TTFM module with the offline guidance of expert trajectories, which are pre-trained and stored in the expert buffer. We consider the expert trajectories (i.e., the performance of stacked *TSOperators* trained on the original dataset $\mathcal{T}$) as the theoretical upper bound for the downstream application

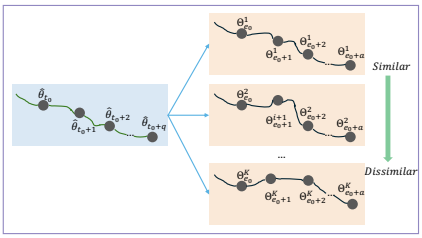


Fig. A.4: Curriculum Trajectory Query and Matching

tasks. Our goal is to obtain a condensed dataset $\mathcal{S}$ that will induce a similar trajectory as that induced by the real training data $\mathcal{T}$ such that models trained on $\mathcal{T}$ and $\mathcal{S}$ achieve similar performance. In addition, we propose a curriculum training trajectory query to perform trajectory matching from similar to dissimilar expert trajectories, which further accelerates the model training. For example, expert trajectories derived from historical stock prices will then guide the alignment of training trajectories extracted from a condensed dataset. This alignment ensures that the condensed dataset's training trajectories capture similar dynamics (such as relationships across different time steps) to those learned from historical data.

In particular, we first train K time series models with the same architecture, i.e., stacked *TSOperators*, denoted as $f_{\mathcal{T}}$, on the original large time series dataset. Then, we can obtain K numbers of expert training trajectories that have a holistic knowledge of the original time series dataset in terms of $f_{\mathcal{T}}$'s training process. We save these model parameters $\{\Theta_{\mathcal{T}}^k\}_{k=1}^K = \{\theta_e^k\}_{e=1}^E$ at certain epoch intervals E in the expert buffer $\mathcal{B}$. Finally, we design a curriculum trajectory query mechanism to sample training trajectories of the expert buffer from similar to dissimilar for trajectory matching. Here, the pre-training of $f_{\mathcal{T}}$ is offline and can be separated from the end-to-end time series condensation, reducing the online computation costs.

Training Trajectory Matching. We use the original dataset to guide the network training, and the parameters trained on this dataset are called an expert trajectory. If a condensed time series dataset is capable of forcing network training dynamics to follow expert trajectories, the idea is that the synthetically trained network will be located close to the model trained on the original time series dataset and will achieve similar test performance for downstream applications.

When sampling a pre-computed parameter trajectory θ_e from the expert buffer $\mathcal{B}$, we aim to minimize the distance between the parameters $\widetilde{\theta}_e$ trained on the condensed dataset and θ_e:

$$\arg\min_{\mathcal{S}} \mathbb{E}_{\theta_e \sim \mathcal{B}}[L_{tmm}(\theta_e|_{e=e_0}^{a}, \widetilde{\theta}_e|_{e=e_0}^{b})], \tag{A.13}$$

where $\mathbb{E}$ is the expectation, $\theta_e|_{e=e_0}^{a}, \widetilde{\theta}_e|_{e=e_0}^{b}$ are the parameters with range $(e_0, e_0 + a)$ and $(e_0, e_0 + b)$, where $e_0 < e_0 + a < E$. More specifically, we

update the condensed TS data according to the trajectory matching loss L_{tmm}.

$$L_{tmm} = \frac{||\widetilde{\theta}_{e_0+b} - \theta_{e_0+a}||_2^2}{||\widetilde{\theta}_{e_0} - \theta_{e_0+a}||_2^2}, \tag{A.14}$$

Note that we initialize the parameters of $f_\theta s$ with those of $f_\mathcal{T}$ at e_0 training step, i.e., $\widetilde{\theta}_{e_0} = \theta_{e_0}$, for more focused training. This way, we align the learning behavior of $f_\mathcal{T}$ with a-steps optimization to $f_\theta s$ with b-steps optimization, imitating the long-term learning behavior of time series modeling.

For the inner loop, we train $f_\theta s$ on the condensed time series data for optimization until the optimal $\widetilde{\theta}^*$. Thus, the objective function of the CT^{2}M is defined as follows.

$$\min_{\mathcal{S}} \mathbb{E}_{\theta_e^* \sim \mathcal{B}}[L_{tmm}(\theta_e|_{e=e_0}^{a}, \widetilde{\theta}_e|_{e=e_0}^{b})] \quad s.t.\ \widetilde{\theta}^{*,\mathcal{S}} = \arg\min_{\widetilde{\theta}} \mathcal{L}(f_{\widetilde{\theta}}, \mathcal{S}), \tag{A.15}$$

where $\mathcal{L}(\cdot)$ is the task-specific loss (e.g., Mean Square Error (MSE)).

Utilizing training trajectory matching, we can reduce computation and memory costs during the condensation process by sampling pre-trained expert trajectories offline. Moreover, long-term trajectory matching provides a more holistic and comprehensive method to imitate the learning behaviors over the original dataset by avoiding that the condensed dataset fits to certain optimization steps short-sightedly.

Curriculum Training Trajectory Query. As shown in Figure A.4, we propose a curriculum training trajectory query method to further enhance the generalization of time series condensation and accelerate the model convergence [36, 37]. Curriculum learning aims to train a model with easy samples first, and then gradually increases the difficulty levels. In our setting, we first query similar training trajectories $\theta_e|_{e=e_0}^{a}$ and $\widetilde{\theta}_e|_{e=e_0}^{b}$, learned on the original and condensed datasets, respectively, to perform trajectory matching. Then, we gradually increase the dissimilarity of the original data training trajectory.

To quantify the similarity between training trajectories of the original dataset $\mathcal{T}$ and the condensed dataset $\mathcal{S}$, we design a minimally interfered retrieval sampling strategy. Specifically, given current parameters $\widetilde{\theta}_{e_0}$ of $f_\theta s$ learned on the condensed dataset, trajectories $\{\Theta_\mathcal{T}^k\}_{k=1}^K$ in buffer $\mathcal{B}$, and a standard task-specific objective function $\min_\theta s\ \mathcal{L}(f_\theta s, \mathcal{S})$, we retrieve trajectories that will be close to $\widetilde{\theta}$ by the update of the foreseen (i.e., future) parameters to select more similar trajectories from $\mathcal{B}$. We update the parameter $\widetilde{\theta}_{e_0}$ for a-steps by gradient matching, as shown in Equation A.16.

$$\widetilde{\theta}_{e_0+a} = \widetilde{\theta}_{e_0} - \sum_{s=1}^{a} (\alpha \nabla \mathcal{L}(f_\theta s, \mathcal{S})), \tag{A.16}$$

where α is the learning rate. Then, we compute the distance $D(\cdot,\cdot)$ between the foreseen trajectory $\widetilde{\theta}|_{e=e_0}^{a}$ of $\mathcal{S}$ and sub-trajectories $\theta^k|_{e=e_0}^{a}$ of $\{\Theta_\mathcal{T}^k\}_{k=1}^K$ in $\mathcal{B}$

Algorithm 7 Curriculum Training Trajectory Query and Matching

Input: A buffer $\mathcal{B}$ with a set of trajectories pre-trained on the original TS dataset $\mathcal{T}$ parameterized by $\{\Theta_{\mathcal{T}}^k\}_{k=1}^K$; current model parameters $\widetilde{\theta}^{\mathcal{S}}$ on $\mathcal{S}$.
Output: Trajectory matching loss L_{tmm}.

1: Distance list $DT \leftarrow []$;
2: Pre-update $\widetilde{\theta}^{\mathcal{S}}$ for a-steps with Equation A.16;
3: $\widetilde{\theta}_{e_0+a} \leftarrow \widetilde{\theta}_{e_0} - \sum_{s=1}^{a}(\alpha \nabla \mathcal{L}(f_{\theta^s}, \mathcal{S}))$;
4: **for** $\Theta_{\mathcal{T}}^k \in \{\Theta_{\mathcal{T}}^k\}_{k=1}^K$ **do**
5: Compute the distance dis_k between $\widetilde{\theta}|_{e_0}^a$ and $\Theta_{\mathcal{T}}^k$ with Equation A.18;
6: $dis_k \leftarrow -D(\widetilde{\theta}|_{e=e_0}^a, \theta^k|_{e=e_0}^a)$;
7: $DT \leftarrow (k, dis_k)$;
8: **end for**
9: Rank DT in a descending order;
10: $\beta \leftarrow 0$
11: **while** $\beta < K$ **do**
12: $k \leftarrow DT[\beta][0]$
13: $L_{tmm} \leftarrow$ Sampling trajectory $\Theta_{\mathcal{T}}^k$ and match training trajectory according to Equation A.14;
14: $\beta \leftarrow \beta + 1$;
15: **end while**
16: **Return** L_{tmm}

as follows.

$$dis_k = -D(\widetilde{\theta}|_{e=e_0}^a, \theta^k|_{e=e_0}^a), \tag{A.17}$$

We use the cosine similarity to measure the distance $D(\cdot)$.

$$D(\widetilde{\theta}|_{e=e_0}^a, \theta^k|_{e=e_0}^a) = \frac{\widetilde{\theta}|_{e=e_0}^a}{||\widetilde{\theta}|_{e=e_0}^a||_2} \cdot \frac{\theta^k|_{e=e_0}^a}{||\theta^k|_{e=e_0}^a||_2} \tag{A.18}$$

Finally, we match current model trajectories of $\mathcal{S}$ with the pre-trained trajectories in descending order based on the similarity $\{dis_k\}_{k=1}^K$.

Algorithm 7 shows the process of the proposed curriculum training trajectory matching. Lines 1–9 concern the curriculum trajectory query, and lines 10–14 concern the trajectory matching. The space and time complexities of Algorithm 7 are $O(K \cdot n)$ and $O((a + K) \cdot n + KlogK)$, respectively.

3.5 Overall Objective Function

The final loss contains three parts: a task-specific loss $\mathcal{L}$, a frequency matching loss L_{Fre}, and a trajectory matching loss L_{tmm}. We combine them together and the overall loss is as follows.

$$L_{all} = \mathcal{L} + L_{Fre} + L_{tmm}. \tag{A.19}$$

The task-specific loss $\mathcal{L}$ is specific to the particular downstream tasks to achieve a better condensed dataset tailored for the intended use. For example, we adopt the cross-entropy loss for task-specific optimization for time series classification tasks to optimize accuracy, while we adopt the mean square error (MSE) loss to minimize the discrepancy between ground truth and predicted values for time series forecasting tasks.

4 Experimental Evaluation

4.1 Experimental Setup

Datasets

Various time series analytics are on edge devices where the storage is limited. We aim at getting a really small condensed dataset. The experiments are carried out on six widely-used time series datasets, covering four application domains: weather, traffic, economics, and energy.

- **Weather.** The Weather dataset contains 21 indicators of weather (e.g., air temperature and humidity), which are collected in Germany. The data is recorded every 10 minutes.
- **Traffic.** The Traffic dataset contains hourly road occupancy rates obtained from sensors located at San Francisco freeways from 2015 to 2016.
- **Electricity.** The Electricity dataset contains the hourly electricity consumption of 321 clients from 2012 to 2014.
- **ETT.** The ETT dataset includes two hourly-level datasets (ETTh1 and ETTh2) and two 15-minute-level datasets (ETTm1 and ETTm2). Each dataset includes 7 oil and load features of electricity transformers between July 2016 and July 2018.

We choose time series forecasting as a representative downstream task, as it is a popular analytics task. In Section 4.4, we also present the performance comparison on the task of time series classification. The numbers of condensed time series are set to 500 and 50 as default for forecasting and classification tasks, respectively. We employ the proposed stacked *TSOperators* as the forecasting and classification models.

Baselines

We compare TimeDC with the following existing methods that include coreset construction methods (i.e., Random, Herding [38], and K-Center [39]), and dataset condensation methods (i.e., DC [10] and MTT [24]).

- **Random.** The Random method randomly selects certain numbers of time series as a coreset.
- **Herding.** The Herding method adds time series observations to coresets greedily [38].
- **K-Center.** The K-Center method first performs K-Center clustering on the original datasets and then chooses observations from each cluster [39].
- **DC.** The DC method employs gradient matching to perform dataset condensation [10].
- **MTT.** The MTT method matches the multi-step training parameters of the condensed data and the original data [24].

Evaluation Metrics

Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) are adopted as the evaluation metrics, which are defined as follows.

$$MAE = \frac{1}{M}\sum_{m=1}^{M}|\hat{\mathcal{Y}}^m - \mathcal{Y}^m|,\ MSE = \frac{1}{M}\sum_{m=1}^{M}||\hat{\mathcal{Y}}^m - \mathcal{Y}^m||^2, \tag{A.20}$$

where M is the testing data size, $\hat{y}^t$ is the prediction and y^t is the ground truth. The smaller the MAE and the RMSE are, the more accurate method is. We also evaluate the efficiency of the models, including the training and dynamic tensor memory cost.

Implementation Details

We implement our model using the Pytorch framework on an NVIDIA GTX 3090 GPU. The hyper-parameters in the model are set as follows. The patch length and stride are set to 16 and 8, respectively. The initial learning rate is 0.0001. The number of *TSOperator* layers is 3. The number of heads in the self-attention layer is set to 16. The number of expert trajectories in the expert buffer is set to 10 by default. ETT datasets and other datasets are split into the training data, validation data, and test data by the ratio of $6:2:2$ and $7:1:2$, respectively. The parameters of the baseline methods are set according to their original papers and any accompanying code. All of the models follow the same experimental setup with prediction length $PL \in \{96, 192, 336\}$ on all datasets.

Table A.1: Overall Performance Comparison on Seven Datasets

Baseline		Random		K-Center		Herding		DC		MTT		TimeDC		Whole Dataset	
Dataset	*PL*	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE
Weather	96	0.731	1.256	0.452	0.687	0.478	0.677	0.361	0.514	0.295	0.244	**0.257**	**0.188**	*0.239*	*0.182*
	192	0.786	1.302	0.487	0.723	0.512	0.688	0.413	0.527	0.344	0.301	**0.285**	**0.247**	*0.261*	*0.195*
	336	0.794	1.311	0.524	0.756	0.554	0.712	0.444	0.567	0.368	0.328	**0.330**	**0.287**	*0.282*	*0.241*
Traffic	96	0.675	1.125	0.503	0.576	0.483	0.554	0.375	0.603	0.279	0.403	**0.254**	**0.375**	*0.247*	*0.337*
	192	0.712	1.144	0.514	0.604	0.517	0.606	0.432	0.633	0.336	0.442	**0.297**	**0.405**	*0.265*	*0.338*
	336	0.729	1.117	0.523	0.611	0.553	0.654	0.449	0.676	0.355	0.471	**0.312**	**0.423**	*0.297*	*0.360*
Electricity	96	0.421	0.669	0.448	0.583	0.501	0.592	0.376	0.513	0.296	0.283	**0.274**	**0.267**	*0.252*	*0.268*
	192	0.450	0.743	0.476	0.601	0.534	0.628	0.419	0.532	0.315	0.337	**0.285**	**0.294**	*0.239*	*0.255*
	336	0.491	0.853	0.506	0.622	0.569	0.477	0.436	0.544	0.339	0.356	**0.304**	**0.322**	*0.271*	*0.285*
ETTh1	96	0.523	0.745	0.554	0.698	0.536	0.656	0.503	0.442	0.456	0.464	**0.413**	**0.401**	*0.354*	*0.386*
	192	0.557	0.786	0.578	0.722	0.589	0.698	0.552	0.508	0.504	0.471	**0.436**	**0.428**	*0.362*	*0.355*
	336	0.588	0.802	0.604	0.745	0.603	0.723	0.556	0.513	0.498	0.464	**0.447**	**0.431**	*0.409*	*0.387*
ETTh2	96	0.487	0.655	0.589	0.711	0.521	0.589	0.463	0.524	0.388	0.342	**0.368**	**0.271**	*0.324*	*0.255*
	192	0.509	0.673	0.605	0.732	0.553	0.621	0.488	0.536	0.416	0.384	**0.389**	**0.302**	*0.332*	*0.257*
	336	0.524	0.689	0.623	0.744	0.564	0.640	0.505	0.540	0.435	0.455	**0.411**	**0.334**	*0.376*	*0.296*
ETTm1	96	0.743	1.124	0.525	0.492	0.607	0.554	0.603	0.665	0.512	0.453	**0.503**	**0.442**	*0.453*	*0.403*
	192	0.764	1.245	0.566	0.510	0.628	0.571	0.597	0.647	0.563	0.501	**0.512**	**0.465**	*0.464*	*0.432*
	336	0.801	1.128	0.571	0.523	0.644	0.582	0.624	0.668	0.552	0.488	**0.500**	**0.483**	*0.477*	*0.455*
ETTm2	96	0.664	0.795	0.486	0.623	0.524	0.558	0.472	0.535	0.376	0.421	**0.354**	**0.391**	*0.347*	*0.381*
	192	0.687	0.804	0.512	0.643	0.549	0.583	0.488	0.567	0.453	0.479	**0.401**	**0.421**	*0.358*	*0.403*
	336	0.702	0.823	0.558	0.661	0.598	0.624	0.493	0.556	0.473	0.523	**0.453**	**0.474**	*0.406*	*0.435*

4.2 Experimental Results

Overall Performance Comparison

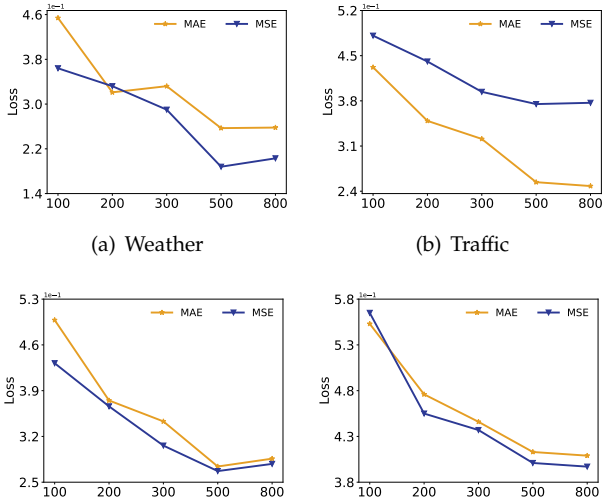
We report the MAE and RMSE values of the methods in Table A.1. The best performance by an existing method (Random, Herding, K-Center, DC, and MTT) is underlined, and the overall best performance is marked in bold. Whole Dataset indicates training on the original dataset and serves as an approximate upper-bound performance; is marked in italics. We use a set of stacked TSFE modules as the basic forecasting model for each baseline. The following observations are made.

- TimeDC achieves the best results on all datasets across all prediction lengths ($PL \in \{96, 192, 336\}$). TimeDC performs better than the best among the baselines by up to 13.49% and 26.59% in terms of MAE and RMSE, respectively. We observe that the performance improvements obtained by TimeDC on the Weather dataset exceed those on the Traffic, Electricity, ETTh1, and ETTh2 in most cases. This is because the Weather has much more training data than the other datasets. TimeDC trained with more training data results in better results. Though the ETTm1 and ETTm2 datasets have substantial training data, TimeDC performs slightly better than baselines on these two datasets. This is because the temporal patterns in ETTm1 and ETTm2 are highly regular and can be captured easily by existing methods.

- The coreset methods (i.e., Random, K-Center, and Herding), perform worse than the condensation methods (i.e., DC, MTT, and TimeDC), where Random has the worst performance in most cases. This is because coreset methods are based on heuristic metrics, that makes it hard to select representative observations and guarantee the optimal solutions.

- A popular condensation method, MTT performs the best among the existing methods except when $PL = 96$ on ETTh1, due to its powerful condensation capability that involves matching multi-step parameters.

The experiment offers evidence that TimeDC is more effective than existing coreset and dataset condensation methods.



(a) Weather (b) Traffic

(c) Electricity (d) ETTh1

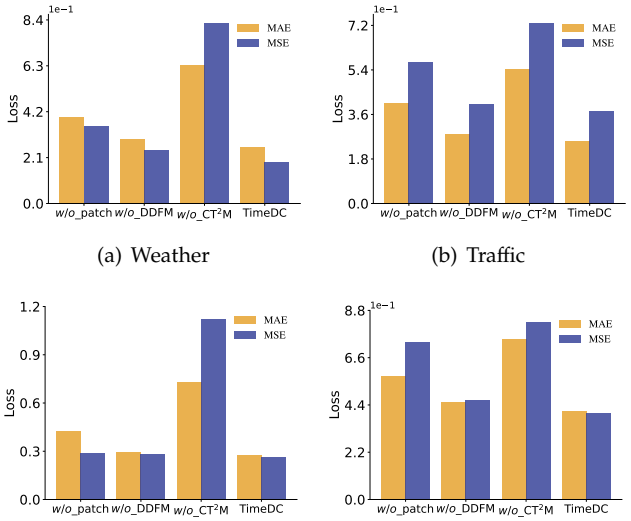
Fig. A.5: Effect of the Size of Condensed TS Dataset on Four Datasets ($PL = 96$)

Effect of the Size of Condensed Time Series Dataset

To study the effect of the size of a condensed time series dataset, we conduct experiments with 100, 200, 300, 500, and 800 condensed time series. The results are shown in Figure A.5. We observe that the curves first drop significantly and then increase slightly (Figures A.5(a) and A.5(c)) or remain almost the same (Figures A.5(b) and A.5(d)). Generally, the results demonstrate that the model performance improves with an increase in the condensed time series data, as more condensed data yields more training data. In addition, it shows that using more condensed time series data for training is more likely to lead to better performance because more useful knowledge is learned from more data. One can also observe that TimeDC with 800 condensed time series performs slightly worse than TimeDC with 500 condensed time series data on the Weather and Electricity. This may be because the patterns in these datasets are relatively simple. Additional condensed time series data might introduce recurring patterns, thereby making the model overfit to these patterns and degrading performance on other data.

Table A.2: Cross-Architecture Performance Comparison

Method	Metric	PL	Weather	Traffic	Electricity	ETT			
						ETTh1	ETTh2	ETTm1	ETTm2
TimeDC	MAE	96	**0.257**	**0.254**	**0.274**	**0.413**	**0.368**	**0.503**	**0.354**
		192	**0.285**	**0.297**	**0.285**	**0.436**	**0.389**	**0.512**	**0.401**
	MSE	96	**0.188**	**0.375**	**0.267**	**0.401**	**0.271**	**0.442**	**0.391**
		192	**0.247**	**0.405**	**0.294**	**0.428**	**0.302**	**0.465**	**0.421**
Autoformer	MAE	96	0.312	0.370	0.343	0.453	0.473	0.548	0.342
		192	0.381	0.385	0.355	0.478	0.491	0.550	0.334
	MSE	96	0.255	0.597	0.236	0.465	0.412	0.542	0.265
		192	0.334	0.613	0.264	0.493	0.488	0.532	0.287
Informer	MAE	96	0.423	0.430	0.428	0.773	0.842	0.576	0.552
		192	0.482	0.476	0.446	0.788	0.954	0.597	0.532
	MSE	96	0.354	0.643	0.253	0.992	1.032	0.624	0.402
		192	0.478	0.710	0.271	0.987	1.055	0.653	0.432
Transformer	MAE	96	0.389	0.412	0.398	0.632	0.506	0.563	0.555
		192	0.588	0.431	0.422	0.612	0.513	0.588	0.576
	MSE	96	0.344	0.578	0.267	0.785	0.579	0.615	0.479
		192	0.524	0.567	0.288	0.732	0.542	0.643	0.455



(a) Weather (b) Traffic

(c) Electricity (d) ETTh1

Fig. A.6: Performance of TimeDC and Its Variants on Four Datasets ($PL = 96$)

Ablation Study

To gain insight into the effects of the different components of TimeDC, including the patching mechanism (patch), decomposition-driven frequency matching (DDFM), and curriculum training trajectory matching (CT^2M), we evaluate three variants:

- ***w/o_Patch***. TimeDC without the patching mechanism.
- ***w/o_DDFM***. TimeDC without the DDFM module but with the CT^2M module.

- ***w/o_CT²M***. TimeDC without the CT²M module of TimeDC but with the DDFM module.

Figure A.6 shows results on Weather, Traffic, Electricity, and ETTh1. Regardless of the datasets, TimeDC outperforms its counterparts without the patching mechanism, the DDFM module, and the CT²M module. This shows that these three components are all useful for effective time series dataset condensation. TimeDC obtains MAE and RMSE reductions by up to 12.88% and 22.95%, respectively, compared with w/o_DDFM. Further, on all datasets, w/o_CT²M performs worst among all variants. TimeDC performs better than w/o_CT²M by at least 44.64% and 48.49% in terms of MAE and MSE, respectively, which indicates the effectiveness of the CT²M module.

Cross-Architecture Performance

Next, we consider the cross-architecture performance of TimeDC. It is important to determine whether the condensed time series data generated by TimeDC can be used to train an unseen network. To assess such cross-architecture performance comprehensively, we consider three representative state-of-the-art network architectures for time series forecasting, including *Autoformer* [33], *Informer* [40], and *Transformer* [41]. We first synthesize condensed time series data with TimeDC and then train these networks with the condensed time series data. The parameters of these network architectures are set based on their original papers and any accompanying code. For brevity, we do not include the detailed parameter settings here but refer to the original papers. For TimeDC, we use a set of stacked TSFE modules as the forecasting network.

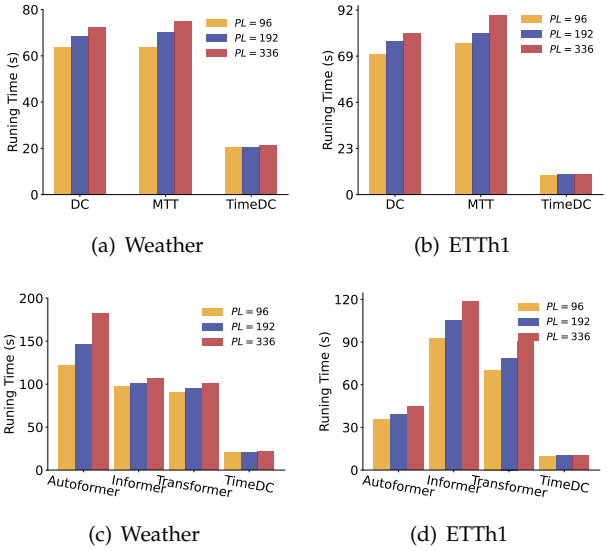
The prediction results are given in Table A.2. Overall, TimeDC has the best performance in most cases, indicating its stable and superior performance, especially on Traffic and Electricity. It also demonstrates the effectiveness of the TSFE module. We observe that Autoformer performs worse than TimeDC but better than Informer and Transformer, while the performances of Informer and Transformer are comparable, especially on ETTm1 and ETTm2. These observations are in line with their performances when trained on the original dataset [33], indicating that TimeDC learns a generalized condensed time series dataset that works across different network architectures. This is because of the powerful feature extraction capabilities of the proposed stacked *TSOperators*, as well as the trajectory matching that imitates the long-term training dynamics of models trained on the original dataset.

Running Efficiency

As resource efficiency is important in dataset condensation to enable scalability, especially on resource-constrained edge computing devices, we study training

Table A.3: Dynamic Tensor Memory Cost on Four Datasets

Dataset	DC	MTT	TimeDC
Weather	10.0 GB	8.9 GB	3.3 GB
Traffic	17.8 GB	13.7 GB	10.9 GB
Electricity	8.5 GB	932.5 MB	516.0 MB
ETTh1	1.9 GB	845.5 MB	280.9 MB



(a) Weather (b) ETTh1

(c) Weather (d) ETTh1

Fig. A.7: Running Efficiency Comparison

time (of an epoch) for the condensation methods. Figures A.7(a) and A.7(b) report the training time on Weather and ETTh1. We see that the training time of TimeDC is much lower than those of DC and MTT, which is largely because of the expert buffer in the CT^{2}M module that stores the pre-computed trajectories. This indicates the feasibility of TimeDC for model deployment in large time series dataset reduction scenarios. We also compare the training time of TimeDC and coreset methods, which are included in the code repository, showing TimeDC has comparable training time with coreset methods. But coreset methods, especially Herding, need more time to construct coreset while having worse performance. Moreover, we compare the memory used by the dynamic (online) tensor across DC, MTT, and TimeDC in Table A.3. TimeDC is able to reduce markedly the online memory and computation costs thanks to the training trajectories precomputed offline.

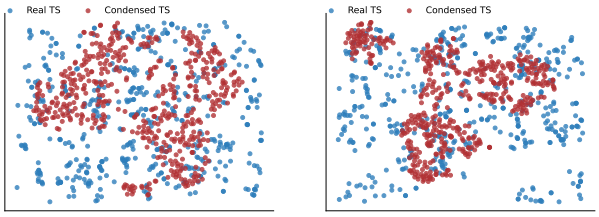
We also study training time across different network architectures on Weather and ETTh1—see Figures A.7(c) and A.7(d). It is clear that TimeDC consumes the least training time. TimeDC is faster than the other methods by at least 73.0%, due to its patching mechanism, which reduces the complexity of the self-attention mechanism through input data simplification. Thus,

Table A.4: Training Time of TimeDC and Training Time on Original Datasets (s/epoch)

Dataset	TimeDC	Original Dataset
Weather	22.39	35.26
Traffic	232.34	346.76
Electricity	314.56	522.85
ETTh1	14.38	20.43

TimeDC has lower training time across different networks, which furthermore yields better performance.

Finally, We compare the training time of TimeDC and the training time on original time series datasets based on the stacked *TSOperators*, as shown in Table A.4. One can see that the training time of TimeDC is significantly lower than when using original datasets, showing the efficiency and practicality of time series dataset condensation. For example, the training time of TimeDC for dataset condensation is reduced by 39.84% compared to using the original dataset on Electricity.



(a) Weather (b) ETTh1

Fig. A.8: Dataset Distribution Comparison on Two Datasets

Case Study on Dataset Distribution

To illustrate the effectiveness of the proposed TimeDC on synthesizing condensed time series datasets that cover the original time series distribution well, we visualize the t-SNE [42] graphs of the original time series dataset and the condensed time series dataset on both Weather and ETTh1. Figure A.8 compares the dataset distributions. Blue and red dots represent the original (i.e., real) time series dataset and the condensed time series dataset, respectively. We sample 500 time series from the original time series dataset for the visualization of the original time series. We observe that the red dots are integrated with the blue dots, indicating that the original time series and condensed time series datasets exhibit similar distributions. This indicates that the condensed dataset is of high quality and that the proposed method is effective. Moreover, the condensed dataset can cover the original dataset evenly on Weather, showing that TimeDC achieves a more diverse and robust condensation result on Weather that exhibits less complicated temporal patterns.

Table A.5: Performance on Streaming Learning ($PL = 96$)

Dataset	Metric	MAE		RMSE	
	Method	Autoformer$_f$	TimeDC	Autoformer$_f$	TimeDC
Weather	$\mathcal{B}_0$	0.603	0.456	0.625	0.477
	$\mathcal{B}_1$	0.694	0.512	0.873	0.533
	$\mathcal{I}$	0.823	0.656	1.014	0.512
Traffic	$\mathcal{B}_0$	0.541	0.465	0.657	0.489
	$\mathcal{B}_1$	0.635	0.502	0.829	0.611
	$\mathcal{I}$	0.735	0.624	0.929	0.784
Electricity	$\mathcal{B}_0$	0.557	0.437	0.601	0.542
	$\mathcal{B}_1$	0.628	0.504	0.875	0.599
	$\mathcal{I}$	0.844	0.689	1.046	0.702
ETTh1	$\mathcal{B}_0$	0.655	0.573	0.812	0.705
	$\mathcal{B}_1$	0.712	0.604	0.933	0.742
	$\mathcal{I}$	1.297	0.762	2.015	0.979

4.3 Application on Streaming Time Series

In real-world scenarios, time series data is often generated incrementally at edge devices that are distributed geographically. It may be preferable to process such streaming time series data on the edge directly to enable compliance with data access restrictions and exploit the potential for more efficient processing and lower latencies. However, time series data keeps growing while the capacities of edge devices are limited. We perform a performance, storage, and parameter comparison to determine whether our condensed time series data can be a key ingredient for streaming time series learning, thereby addressing also the catastrophic forgetting problem [9, 43], due to its condensed nature.

Performance Comparison

A naive solution to streaming time series learning is to use only newly arrived data, not all the data collected so far, to update model parameters continuously, called fine-tuning. We compare the performance of TimeDC and fine-tuning to test the effectiveness of TimeDC at streaming learning. For the fine-tuning, we use Autoformer as the basic model, entitled Autoformer$_f$. We split the original dataset into a base set $\mathcal{B}$ and an incremental set $\mathcal{I}$ with the ratio $7:3$. $\mathcal{B}$ and $\mathcal{I}$ are further split into training data, validation data, and test data with the ratios $7:1:2$. We first train and test the model on $\mathcal{B}$. The result is denoted as $\mathcal{B}_0$. Then, we update the learned model with $\mathcal{I}$ and test on $\mathcal{B}$ and $\mathcal{I}$ simultaneously, with the results denoted as $\mathcal{B}_1$ and $\mathcal{I}$, respectively. For the model update stage of TimeDC, we condense the base set and incorporate the condensed data into $\mathcal{I}$ for subsequent model training. The results in Table A.5 show that TimeDC has the best performance in terms of MAE and

RMSE when compared with Autoformer$_f$ on four datasets. For example, TimeDC outperforms Autoformer$_f$ by 18.36% ~ 21.54% and 9.82% ~ 32.89% in terms of MAE and RMSE on Electricity, respectively. Autoformer$_f$ provides acceptable MAE and RMSE results on the base set, but their performance deteriorates on incremental set, especially on ETTh1. This indicates that a simple fine-tuning method is insufficient, due to forgetting problems caused by the possible concept drift. TimeDC achieves relatively stable performance on all base sets and incremental sets, demonstrating its superiority.

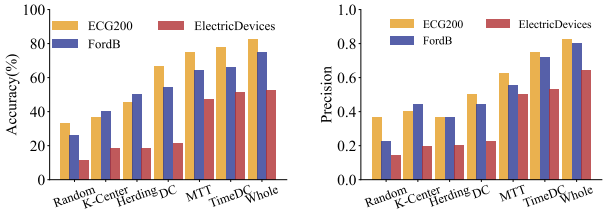
Table A.6: Storage Comparison on Four Datasets

Storage	Weather	Traffic	Electricity	ETTh1
Whole Dataset	2.9 GB	20.0 GB	11.2 GB	313.1 MB
Condensed TS	38.5 MB	827.5 MB	308.2 MB	12.8 MB

Storage and Parameter Comparison

As storage size is a main concern of time series streaming learning on edge devices, we compare the storage cost of the pre-processed whole dataset and the condensed time series dataset using four datasets, as shown in Table A.6. Generally, the condensed time series data of TimeDC reduces the storage space substantially compared with the whole dataset. The condensed time series data is only 4.09% of the whole dataset on ETTh1. Thus, TimeDC enables much lower storage cost while achieving more promising performance on time series streaming learning.

Another limitation of edge devices is limited computational capabilities. We thus compare the number of parameters of different network architectures including TimeDC (i.e., TSFE), Autoformer, Informer, and Transformer—see Table A.7. TimeDC achieves much fewer parameters, indicating that the TSFE module of TimeDC (mainly the patching mechanism) can significantly reduce computational costs. Tables A.5, A.6, and A.7 indicate the feasibility and scalability of TimeDC for streaming learning on edge devices.



(a) Accuracy (b) Precision

Fig. A.9: Time Series Classification Performance

Table A.7: Parameter Comparison on Four Architectures

Dataset	Autoformer	Informer	Transformer	TimeDC
Weather	10.6 M	11.4 M	10.6 M	1.8 M
Etth1	10.5	11.3 M	10.5 M	0.2 M

4.4 Performance on Time Series Classification

It is important to show the generalization of the proposed TimeDC that can be used in other time series tasks. We conduct experiments on time series classification. The experiments are carried out on three datasets (i.e., ECG200, ElectricDevices, and FordB) of the UCR time series archive[1] [44]. A set of stacked TSFE modules with an FC is used as the basic classification model. CrossEntropy Loss is used as the objective of time series classification. *Accuracy* and *Recall* are adopted as evaluation metrics. The number of condensed time series is set to 50 for each dataset. The overall performance comparison results are provided in Figure A.9. The whole (dataset) indicates training on the whole original dataset, serving as an approximate upper-bound performance. TimeDC performs better than the best among the baselines by up to 10.40% and 67.31% in terms of *Accuracy* and *Precision*, respectively. All coreset construction methods perform worse than dataset condensation methods. Herding has the best performance in most cases among coreset methods. Overall, TimeDC achieves the best results on three time series classification datasets among baselines, which shows the feasibility of TimeDC that can be extended to other time series tasks.

5 Related work

5.1 Time Series Modeling

Time series modeling attracts increasing interest due to the growing availability of time series data and rich downstream applications, such as traffic prediction [3, 45, 46], electricity prediction [35, 40], and anomaly detection [7, 47, 48]. Traditional time series analytic models are mostly based on statistical models [6, 45, 49]. However, the statistical models cannot capture complex temporal correlations of time series data effectively due to their limited learning capacity. Recent advances in deep learning techniques have sparked a surge of interest in applying neural network architectures for time series modeling [3, 28, 40] outperforming traditional statistical models, including temporal convolutional network (TCN) based methods [50, 51], recurrent neural network (RNN) based methods [52, 53], and transformer based methods [4, 8, 40]. However, these methods are mostly supervised, and large training data is required resulting

[1] https://www.cs.ucr.edu/~eamonn/time_series_data_2018/

in high computational cost. At such scales, it becomes burdensome to store and preprocess the data and calls for specialized equipment and infrastructure to train machine learning models on them.

5.2 Coreset and Dataset Condensation

Coreset construction [9, 13, 14, 18, 54, 55] is the traditional dataset reduction approach that works by identifying the most representative training samples in an original dataset, aiming at achieving models trained on the coreset that are provably competitive with those built on the full dataset. Typically, coreset construction methods choose samples that are representative for training based on heuristic criteria [9, 13, 14, 18], e.g., minimizing distance between coreset and whole-dataset centers [13], Nonetheless, these heuristic methods cannot guarantee optimal solutions and the presence of representative samples for the downstream task.

A recent approach, dataset condensation (or distillation) [10, 19, 23, 24], is proposed to address these limitations by learning a small typical dataset that distills the most important knowledge from a given large dataset, such that a model trained on it can obtain comparable testing accuracy to that trained on the original training set. Existing dataset distillation methods have demonstrated superior performance, which can be categorized into matching-based methods [10, 20, 24] and kernel-based methods [30, 56]. Matching-based methods generate synthetic datasets by matching gradients [10], multi-step parameters [24], and distributions [20] between two surrogate models trained on the synthetic dataset and the original dataset. Kernel-based methods [30, 56] treat the synthetic dataset as the parameters to be optimized inspired by kernel functions e.g., neural tangent kernel. However, most of the above methods are designed for computer vision, and cannot be applied to time series directly due to complex temporal correlations such as seasonality and trend. In addition, these methods require substantial storage costs.

6 Conclusion

We present TimeDC, a new efficient time series dataset condensation framework that endeavors to synthesize a small but informative condensed time series dataset summarizing the original large time series dataset. To capture the complex temporal dependencies, we design a time series feature extraction module with stacked *TSOperators*. In addition, decomposition-driven frequency matching is proposed to ensure similar temporal patterns between the condensed and original time series datasets. To enable effective and generalized dataset condensation, we feature a curriculum training trajectory matching module with an expert buffer that aims to decrease the training cost.

Comprehensive experiments on original datasets offer evidence that TimeDC achieves state-of-the-art accuracy and requires fewer computational and storage resources.

References

[1] S. K. Jensen, T. B. Pedersen, and C. Thomsen, "Modelardb: Modular model-based time series management with spark and cassandra," *PVLDB*, vol. 11, no. 11, pp. 1688–1701, 2018.

[2] C. Wang, X. Huang, J. Qiao, T. Jiang, L. Rui, J. Zhang, R. Kang, J. Feinauer, K. A. McGrail, P. Wang *et al.*, "Apache iotdb: Time-series database for internet of things," *PVLDB*, vol. 13, no. 12, pp. 2901–2904, 2020.

[3] X. Wu, D. Zhang, M. Zhang, C. Guo, B. Yang, and C. S. Jensen, "AutoCTS+: Joint neural architecture and hyperparameter search for correlated time series forecasting," *SIGMOD*, vol. 1, no. 1, pp. 1–26, 2023.

[4] S. Tuli, G. Casale, and N. R. Jennings, "Tranad: deep transformer networks for anomaly detection in multivariate time series data," *PVLDB*, vol. 15, no. 6, pp. 1201–1214, 2022.

[5] L. Rui, X. Huang, S. Song, Y. Kang, C. Wang, and J. Wang, "Time series representation for visualization in apache iotdb," *SIGMOD*, vol. 2, no. 1, pp. 1–26, 2024.

[6] S. Basu and M. Meckesheimer, "Automatic outlier detection for time series: an application to sensor data," *KAIS*, vol. 11, pp. 137–154, 2007.

[7] D. Campos, M. Zhang, B. Yang, T. Kieu, C. Guo, and C. S. Jensen, "Lightts: Lightweight time series classification with adaptive ensemble distillation," *SIGMOD*, vol. 1, no. 2, pp. 1–27, 2023.

[8] T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin, "Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting," in *ICML*, 2022, pp. 27 268–27 286.

[9] Y. Li, Y. Shen, and L. Chen, "Camel: Managing data for efficient stream learning," in *SIGMOD*, 2022, pp. 1271–1285.

[10] B. Zhao, K. R. Mopuri, and H. Bilen, "Dataset condensation with gradient matching," in *ICLR*, 2021. [Online]. Available: https://openreview.net/forum?id=mSAKhLYLSsl

[11] X. Yu, Y. Peng, F. Li, S. Wang, X. Shen, H. Mai, and Y. Xie, "Two-level data compression using machine learning in time series database," in *ICDE*, 2020, pp. 1333–1344.

[12] J. Xiao, Y. Huang, C. Hu, S. Song, X. Huang, and J. Wang, "Time series data encoding for efficient storage: A comparative analysis in apache iotdb," *PVLDB*, vol. 15, no. 10, pp. 2148–2160, 2022.

[13] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan, "Approximating extent measures of points," *JACM*, vol. 51, no. 4, pp. 606–635, 2004.

[14] C. Chai, J. Liu, N. Tang, J. Fan, D. Miao, J. Wang, Y. Luo, and G. Li, "Goodcore: Data-effective and data-efficient machine learning through coreset selection over incomplete data," *SIGMOD*, vol. 1, no. 2, pp. 1–27, 2023.

[15] S. Hasani, S. Thirumuruganathan, A. Asudeh, N. Koudas, and G. Das, "Efficient construction of approximate ad-hoc ml models through materialization and reuse," *PVLDB*, vol. 11, no. 11, pp. 1468–1481, 2018.

[16] D. Feldman and M. Langberg, "A unified framework for approximating and clustering data," in *STOC*, 2011, pp. 569–578.

[17] M. Lucic, M. Faulkner, A. Krause, and D. Feldman, "Training gaussian mixture models at scale via coresets," *J. Mach. Learn. Res*, vol. 18, no. 160, pp. 1–25, 2018.

[18] S. Gupta, R. Kumar, K. Lu, B. Moseley, and S. Vassilvitskii, "Local search methods for k-means with outliers," *PVLDB*, vol. 10, no. 7, pp. 757–768, 2017.

[19] T. Wang, J. Zhu, A. Torralba, and A. A. Efros, "Dataset distillation," *CoRR*, vol. abs/1811.10959, 2018.

[20] G. Zhao, G. Li, Y. Qin, and Y. Yu, "Improved distribution matching for dataset condensation," in *CVPR*, 2023, pp. 7856–7865.

[21] C. C. Aggarwal and P. S. Yu, "On static and dynamic methods for condensation-based privacy-preserving data mining," *TODS*, vol. 33, no. 1, pp. 1–39, 2008.

[22] F. Angiulli, "Fast nearest neighbor condensation for large data sets classification," *TKDE*, vol. 19, no. 11, pp. 1450–1464, 2007.

[23] S. Lei and D. Tao, "A comprehensive survey of dataset distillation," *TPAMI*, vol. 46, no. 1, pp. 17–32, 2024.

[24] G. Cazenavette, T. Wang, A. Torralba, A. A. Efros, and J.-Y. Zhu, "Dataset distillation by matching training trajectories," in *CVPR*, 2022, pp. 4750–4759.

[25] H. Miao, J. Shen, J. Cao, J. Xia, and S. Wang, "Mba-stnet: Bayes-enhanced discriminative multi-task learning for flow prediction," *TKDE*, vol. 35, no. 7, pp. 7164–7177, 2023.

[26] Q. Wen, K. He, L. Sun, Y. Zhang, M. Ke, and H. Xu, "Robustperiod: Robust time-frequency mining for multiple periodicity detection," in *SIGMOD*, 2021, pp. 2328–2337.

[27] Y. Yao, D. Li, H. Jie, H. Jie, T. Li, J. Chen, J. Wang, F. Li, and Y. Gao, "Simplets: An efficient and universal model selection framework for time series forecasting," *PVLDB*, vol. 16, no. 12, pp. 3741–3753, 2023.

[28] A. Bonifati, F. D. Buono, F. Guerra, and D. Tiano, "Time2feat: learning interpretable representations for multivariate time series clustering," *PVLDB*, vol. 16, no. 2, pp. 193–201, 2022.

[29] S. Wang, M. Zhang, H. Miao, Z. Peng, and P. S. Yu, "Multivariate correlation-aware spatio-temporal graph convolutional networks for multi-scale traffic prediction," *TIST*, vol. 13, no. 3, pp. 1–22, 2022.

[30] Y. Zhou, E. Nezhadarya, and J. Ba, "Dataset distillation using neural feature regression," *NeurIPS*, vol. 35, pp. 9813–9827, 2022.

[31] K. S. Tai, V. Sharan, P. Bailis, and G. Valiant, "Sketching linear classifiers over data streams," in *SIGMOD*, 2018, pp. 757–772.

[32] R. B. Cleveland, W. S. Cleveland, J. E. McRae, and I. Terpenning, "Stl: A seasonal-trend decomposition," *J. Off. Stat*, vol. 6, no. 1, pp. 3–73, 1990.

[33] H. Wu, J. Xu, J. Wang, and M. Long, "Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting," *NeurIPS*, vol. 34, pp. 22 419–22 430, 2021.

[34] C. Chiarella, X.-Z. He, and C. Hommes, "A dynamic analysis of moving average rules," *Journal of Economic Dynamics and Control*, vol. 30, no. 9-10, pp. 1729–1753, 2006.

[35] Y. Nie, N. H. Nguyen, P. Sinthong, and J. Kalagnanam, "A time series is worth 64 words: Long-term forecasting with transformers," in *ICLR*, 2023.

[36] K. Zheng, G. Chen, M. Herschel, K. Y. Ngiam, B. C. Ooi, and J. Gao, "Pace: learning effective task decomposition for human-in-the-loop healthcare delivery," in *SIGMOD*, 2021, pp. 2156–2168.

[37] Q. Wang, S. Whitmarsh, V. Navarro, and T. Palpanas, "iedeal: A deep learning framework for detecting highly imbalanced interictal epileptiform discharges," *PVLDB*, vol. 16, no. 3, pp. 480–490, 2022.

[38] M. Welling, "Herding dynamical weights to learn," in *ICML*, 2009, pp. 1121–1128.

[39] R. Z. Farahani and M. Hekmatfar, *Facility location: concepts, models, algorithms and case studies*. Springer Science & Business Media, 2009.

[40] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, "Informer: Beyond efficient transformer for long sequence time-series forecasting," in *AAAI*, vol. 35, no. 12, 2021, pp. 11 106–11 115.

[41] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *NeurIPS*, vol. 30, 2017.

[42] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *J Mach Learn Res*, vol. 9, no. 11, 2008.

[43] Z. Chu, R. Li, S. Rathbun, and S. Li, "Continual causal inference with incremental observational data," in *ICDE*, 2023, pp. 3430–3439.

[44] H. A. Dau, A. Bagnall, K. Kamgar, C.-C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, and E. Keogh, "The ucr time series archive," *JAS*, vol. 6, no. 6, pp. 1293–1305, 2019.

[45] S. Shekhar and B. M. Williams, "Adaptive seasonal time series models for forecasting short-term traffic flow," *TRR*, vol. 2024, no. 1, pp. 116–125, 2007.

[46] Z. Fang, L. Pan, L. Chen, Y. Du, and Y. Gao, "Mdtp: A multi-source deep traffic prediction framework over spatio-temporal trajectory data," *PVLDB*, vol. 14, no. 8, pp. 1289–1297, 2021.

[47] J. Paparrizos, Y. Kang, P. Boniol, R. S. Tsay, T. Palpanas, and M. J. Franklin, "Tsb-uad: an end-to-end benchmark suite for univariate time-series anomaly detection," *PVLDB*, vol. 15, no. 8, pp. 1697–1711, 2022.

[48] E. Sylligardos, P. Boniol, J. Paparrizos, P. Trahanias, and T. Palpanas, "Choose wisely: An extensive evaluation of model selection for anomaly detection in time series," *PVLDB*, vol. 16, no. 11, pp. 3418–3432, 2023.

[49] M. Lippi, M. Bertini, and P. Frasconi, "Short-term traffic flow forecasting: An experimental comparison of time-series analysis and supervised learning," *TITS*, vol. 14, no. 2, pp. 871–882, 2013.

[50] C. H. Liu, C. Piao, X. Ma, Y. Yuan, J. Tang, G. Wang, and K. K. Leung, "Modeling citywide crowd flows using attentive convolutional lstm," in *ICDE*, 2021, pp. 217–228.

[51] Y. Cheng, P. Chen, C. Guo, K. Zhao, Q. Wen, B. Yang, and C. S. Jensen, "Weakly guided adaptation for robust time series forecasting," *PVLDB*, vol. 17, no. 4, pp. 766–779, 2024.

[52] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski, "Deepar: Probabilistic forecasting with autoregressive recurrent networks," *Int J Forecast*, vol. 36, no. 3, pp. 1181–1191, 2020.

[53] H. Yuan, G. Li, Z. Bao, and L. Feng, "Effective travel time estimation: When historical trajectories over road networks matter," in *SIGMOD*, 2020, pp. 2135–2149.

[54] D. Feldman, M. Schmidt, and C. Sohler, "Turning big data into tiny data: Constant-size coresets for k-means, pca, and projective clustering," *SICOMP*, vol. 49, no. 3, pp. 601–657, 2020.

[55] Q. Zhang, J. Liu, and W. Wang, "Approximate clustering on distributed data streams," in *ICDE*, 2008, pp. 1131–1139.

[56] T. Nguyen, Z. Chen, and J. Lee, "Dataset meta-learning from kernel ridge-regression," in *ICLR*, 2020.

Paper B

A Unified Replay-based Continuous Learning Framework for Spatio-Temporal Prediction on Streaming Data

Hao Miao, Yan Zhao, Chenjuan Guo, Bin Yang, Kai Zheng,
Feiteng Huang, Jiandong Xie, Christian S. Jensen

The paper has been published in the
40th IEEE International Conference on Data Engineering, 2024.

© 201X IEEE
The layout has been revised.

Abstract

The widespread deployment of wireless and mobile devices results in a proliferation of spatio-temporal data that is used in applications, e.g., traffic prediction, human mobility mining, and air quality prediction, where spatio-temporal prediction is often essential to enable safety, predictability, or reliability. Many recent proposals that target deep learning for spatio-temporal prediction suffer from so-called catastrophic forgetting, where previously learned knowledge is entirely forgotten when new data arrives. Such proposals may experience deteriorating prediction performance when applied in settings where data streams into the system. To enable spatio-temporal prediction on streaming data, we propose a unified replay-based continuous learning framework. The framework includes a replay buffer of previously learned samples that are fused with training data using a spatio-temporal mixup mechanism in order to preserve historical knowledge effectively, thus avoiding catastrophic forgetting. To enable holistic representation preservation, the framework also integrates a general spatio-temporal autoencoder with a carefully designed spatio-temporal simple siamese (STSimSiam) network that aims to ensure prediction accuracy and avoid holistic feature loss by means of mutual information maximization. The framework further encompasses five spatio-temporal data augmentation methods to enhance the performance of STSimSiam. Extensive experiments on real data offer insight into the effectiveness of the proposed framework.

1 Introduction

The continued digitization of societal processes and the accompanying deployment of sensing technologies generate increasingly massive amounts of spatio-temporal data. For example, populations of in-road sensors provide data that captures traffic flow in multiple locations across time. Further, applications increasingly ingest large amounts of spatio-temporal data that is being generated continuously, which is called streaming spatio-temporal data. The data from in-road sensors is an example of such data.

In this study, we focus on the novel problem of prediction on streaming spatio-temporal data, where the general goal is to learn a model from streaming spatio-temporal data with spatial and temporal correlations while preserving learned historical knowledge and capturing spatio-temporal patterns to accurately predict future spatio-temporal observations.

Many spatio-temporal prediction applications, e.g., traffic flow prediction [1–3], traffic speed prediction [4, 5] and on-demand service prediction [6], exist along with accompanying, well-customized prediction models. These models use a variety of means to predict spatio-temporal data, including approaches based on traditional statistics [7, 8] and convolutional [9–11] and recurrent [4, 12] neural networks. However, existing models are trained stat-

ically and fail to handle streaming data. Static models are often trained once to fit a particular dataset and then make predictions that aim to fit another dataset with the underlying assumption that the two datasets follow the same distribution. However, concept drift, meaning that data distributions change over time, occurs often in streaming spatio-temporal data. Thus, direct application of static models to streaming data may cause remarkable performance degradation [13].

As a result, we need a new kind of continuous learning (CL) model that can adapt continually to the data it sees and can keep on learning spatio-temporal prediction tasks over time. However, it is non-trivial to develop this kind of model, due to the following challenges.

Challenge I: catastrophic forgetting. It is challenging to alleviate catastrophic forgetting in CL for spatio-temporal prediction. Catastrophic forgetting is the tendency to abruptly forget previously learned knowledge, which occurs when static models are simply retrained using newly arrived data [13]. When a model is retrained continuously based on incoming data, the distributions of which are different (due to concept drift), prediction performance on preceding tasks deteriorates [14]. This is because the model is always retrained on new data obtained during one period and is then used to predict for another period. When the data arrives continuously, the knowledge learned by the model keeps drifting. Although many efforts have been made to use CL techniques to address catastrophic forgetting in computer vision [15, 16] and natural language processing [17], these techniques cannot be applied to spatio-temporal prediction directly due to the unique characteristics of spatio-temporal data. Specifically, no CL model exists that can effectively capture spatial dependencies or temporal correlations in streaming spatio-temporal data.

Challenge II: diversity of spatio-temporal data and prediction applications. It remains a key problem to discover the commonalities of diverse spatio-temporal data and prediction applications. Although various well-customized static models exist, it is time-consuming and unrealistic to convert each such model to its continuous version since there are different specific settings (e.g., network architectures, datasets, and objective functions) to consider for each spatio-temporal prediction model. It is thus highly desirable but also non-trivial to achieve a model that is accurate and efficient across many streaming spatio-temporal prediction applications.

Challenge III: holistic feature preservation. It is challenging for existing models to learn holistic features for spatio-temporal prediction on streaming data. Specifically, holistic features preserve semantic similarities across multiple time periods. In continuous spatio-temporal prediction, preserving previously learned semantic features may facilitate future prediction. For example, traffic patterns learned during off-peak hours on previous weekdays may be helpful for prediction of subsequent weekdays. Most existing CL models focus on

learning discriminative features for a current task, while ignoring previously learned features that may be useful for future tasks [18], which often lead to unsatisfactory results for future prediction and thus affect continuous spatio-temporal prediction adversely.

This study addresses the above challenges by providing a Unified Replay-based Continuous Learning (URCL) framework for spatio-temporal prediction on streaming data. URCL encompasses three main modules: data integration, spatio-temporal continuous representation learning (STCRL), and spatio-temporal prediction. To alleviate catastrophic forgetting (Challenge I), we propose a spatio-temporal mixup (STMixup) mechanism to fuse current sampled spatio-temporal observations with selected samples from a replay buffer that stores a subset of previously learned observations. We also propose a ranking-based maximally interfered retrieval sampling strategy to select representative samples from the buffer.

To support diverse spatio-temporal data and prediction applications (Challenge II), we discover the commonalities of existing methods that are typically based on an autoencoder architecture. We propose a novel spatio-temporal prediction network including a spatio-temporal encoder (STEncoder) and a spatio-temporal decoder (STDecoder) to capture complex spatio-temporal correlations to enable accuracy.

To address the issue of holistic-feature preservation (Challenge III), we propose a spatio-temporal simple siamese (STSimSiam) network to avoid holistic feature loss in the STCRL module. Specifically, the STSimSiam network contains two STEncoders and a projection multi-layer perceptron (MLP) head, where the STEncoders are shared with that of the spatio-temporal prediction network. We first use mutual information maximization to ensure holistic feature preservation in streaming spatio-temporal prediction. In addition, considering that data augmentation can help the model learn more effective representations [19, 20], we provide five spatio-temporal data augmentation methods based on an exploration of the exclusive characteristics of spatio-temporal data, thus achieving effective holistic spatio-temporal feature learning.

The major contributions are summarized as follows.

- To the best of our knowledge, this is the first study to systematically investigate continuous learning for spatio-temporal prediction on streaming data. We propose a unified replay-based continuous learning framework entitled URCL for spatio-temporal prediction on streaming data, where a replay buffer and a spatio-temporal mixup mechanism are designed to alleviate catastrophic forgetting.

- To contend with the diversity of spatio-temporal data and prediction applications, we design a spatio-temporal autoencoder including an STEn-

coder and an STDecoder for effective spatio-temporal feature learning and prediction.

- A novel STSimSiam network is designed to make learned features in latent spaces more holistic, by applying mutual information maximization to preserve features for continuous spatio-temporal prediction. We also provide five data augmentation methods by considering unique spatiotemporal properties.
- We report on experiments using real datasets, offering evidence of the effectiveness of the proposed URCL.

The remainder of this paper is organized as follows. Section 2 surveys the related work, and Section 3 introduces preliminary concepts and the streaming spatio-temporal prediction problem. We then present the URCL framework in Section 4, followed by the experimental results in Section 5. Section 6 concludes the paper.

2 Related work

We briefly review prior studies on spatio-temporal data prediction and continuous learning.

2.1 Spatio-Temporal Data Prediction

Spatio-temporal data prediction attracts increasing interest due to the increasing availability of spatio-temporal data and rich applications, such as traffic prediction [1, 21–23], precipitation prediction [24], and air quality prediction [25]. Traditional spatio-temporal prediction models are mostly based on statistical models [8, 26]. However, the statistical models cannot capture complex spatial and temporal correlations of spatio-temporal data effectively due to their limited learning capacity.

With the advance of deep learning techniques, various deep learning based methods address spatio-temporal data prediction [4, 27–35], and outperform traditional statistical models. One line of study [1, 2] treats the spatio-temporal data of an entire city as images and applies convolutional neural network (CNN) to extract spatial correlations. Another line of study [4, 9, 12, 36–39] employs graph neural network to perform spatio-temporal prediction by modeling global spatial dependencies and local spatial correlations effectively. However, these methods cannot support stream setting and suffer from catastrophic forgetting, which makes continuous or lifelong learning difficult for spatio-temporal prediction.

2.2 Continuous Learning

Continuous learning, which is also called lifelong or incremental learning, learns a sequence of tasks incrementally with knowledge transfer and without catastrophic forgetting [40]. The goal of continuous learning is to extend knowledge acquired gradually from an infinite data stream and use it for future learning.

In the early stage, continuous learning studies target the domain of object recognition [41, 42]. Thrun et al. [41] propose several lifelong learning algorithms encompassing memory-based and neural network-based approaches. Ruvolo et al. [42] design an Efficient Lifelong Learning Algorithm for online multi-task learning.

With the rapid development of deep learning techniques, focus is on deep learning based continuous learning methods [15, 16], which can be divided into replay-based, regularization-based, and architecture-based methods. Specifically, replay-based methods [16] usually adopt an explicit buffer to store a subset of training samples or learn a generator. Regularization-based methods include a regularization term in the loss function [15, 43]. In architecture-based methods [13], a different sub-network is dedicated to each incremental learning task. However, most of the above methods are designed for computer vision and natural language processing, and cannot be applied to spatio-temporal data prediction directly due to complex spatio-temporal patterns and unique spatio-temporal semantics.

Although previous studies [44, 45] predict streaming traffic flow, their problem settings differ substantially from ours that focusing on node incremental learning where the number of traffic sensors varies across time. In our work, the number of sensors does not vary, while the instances vary across time.

3 Problem Statement

We proceed to present necessary preliminaries and then define the problem addressed.

Advances in hardware and wireless network technologies have resulted in multi-functional sensor devices [46]. This development enables systems, called sensor networks, consisting of tiny sensor nodes spread across large geographical areas, recording streaming spatio-temporal data.

Definition 22 (Sensor Network) *A sensor network is denoted by a graph $G = (V, E)$, where V is a sensor node set, and E is an edge set. Each node $v_i \in V$ represents a sensor, and each edge $e_{i,j} \in E$ indicates connectivity between sensors v_i and v_j.*

Definition 23 (Spatio-temporal Observation) *Given a sensor network G, a spatio-temporal observation collected by all sensors at time slot t (e.g., 9:00 a.m.–9:15 a.m.)*

with a sampling interval Δt (e.g., 15 minutes) is denoted by $X_t \in \mathcal{R}^{|V| \times C}$, where $|V|$ denotes the number of sensors and C is the dimensionality of node features (e.g., traffic volume and speed).

In the rest of the paper, we will use *observation* place of *spatio-temporal observation* when meaning is clear from the context.

Definition 24 (Streaming Spatio-temporal Data Sequence) *Given a sensor network G and a time period $\mathbb{T}_i$ containing n consecutive time slots, i.e., $\mathbb{T}_i = \left\langle t_i^1, t_i^2, \cdots, t_i^n \right\rangle$, a streaming spatio-temporal data sequence is a sequence of matrices, each representing an observation at a specific time slot t_i^j with one sampling interval, where $t_i^{j+1} - t_i^j = \Delta t (1 \leq j \leq n-1)$. In particular, a streaming spatio-temporal data sequence D_i is a sequence of observations $D_i = \left\langle X_{t_i^1}, X_{t_i^2}, \cdots, X_{t_i^n} \right\rangle$, where n is also the sequence length.*

Based on the above definitions, we formally define the **Streaming Spatio-Temporal Prediction (SSTP)** problem as follows.

SSTP Problem. Consider a sensor network G that emits a sequence $\mathbb{D} = \langle D_1, D_2, \cdots, D_m \rangle$ $(m \geq 1)$ of streaming spatio-temporal data sequences, where D_i denotes a sequence of observations during time period $\mathbb{T}_i$ and $|D_i| = n$ (e.g., $\mathbb{T}_i$ is a day, and n is 96 when having 15min interval). Given a current observation $X_{t_i^j}$ that is contained in D_i $(1 \leq i \leq m)$, the SSTP problem aims to learn a function $f_i(\cdot)$ to predict N future observations based on the current observation and its previous $M-1$ $(M < n)$ historical observations while maximally preserving the learned knowledge from previous streaming data sequences $\langle D_1, \cdots, D_{i-1} \rangle$, i.e.,

$$[\overbrace{\cdots, X_{t_i^{j-1}}, X_{t_i^j}}^{M \text{ observations}}; G] \xrightarrow{f_i(\cdot)} [\overbrace{X_{t_i^{j+1}}, X_{t_i^{j+2}, \ldots}}^{N \text{ observations}}], \tag{B.1}$$

where the learned knowledge from $\langle D_1, \cdots, D_{i-1} \rangle$ is preserved maximally.

4 Methodology

We propose a framework, namely Unified Replay-based Continuous Learning (URCL), for spatio-temporal prediction on streaming data. We first give an overview of the framework and then provide specifics on each module in the framework.

4.1 Framework Overview

The framework consists of three major modules: data integration, spatio-temporal continuous representation learning (STCRL) and spatio-temporal

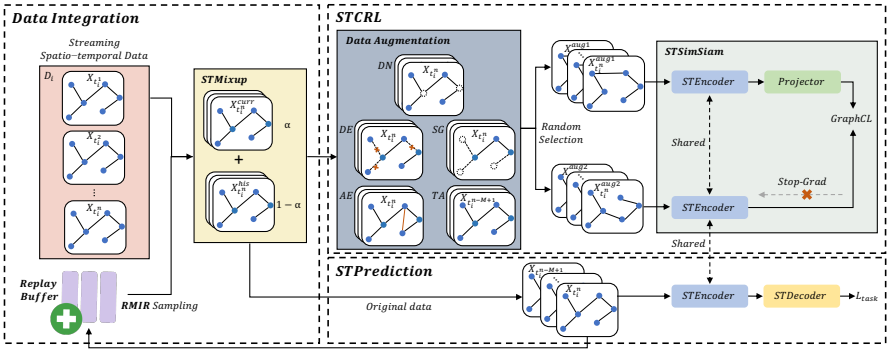


Fig. B.1: URCL Framework Overview

prediction (STPrediction), as shown in Figure B.1.

Data Integration. Considering that we feed data into the framework in a stream, we first sample data from the current dataset D_i and sample historical data from a replay buffer, which stores a subset of previously learned observations, by a ranking-based maximally interfered retrieval (RMIR) sampling strategy. Then we integrate the two kinds of data with the help of the proposed spatio-temporal mixup (STMixup) mechanism, to goal being to better accumulate the spatio-temporal knowledge and alleviate catastrophic forgetting.

Spatio-Temporal Continuous Representation Learning (STCRL). In this module, a spatio-temporal SimSiam (STSimSiam) network, which is a variant of self-supervised Siamese networks [19], is adopted for holistic representation learning by means of mutual information maximization. For the assistance of self-supervised learning, we propose five different data augmentation methods, i.e., DropNodes (DN), DeleteEdges (DE), SubGraph (SG), AddEdge (AE), and TimeShifting (TS), based on the specific characteristics of spatio-temporal data. Then, the augmented data generated by two randomly selected augmentation methods are inserted into two spatio-temporal encoders (STEncoders) that share parameters with each other, followed by a projection MLP (projector).

Spatio-Temporal Prediction (STPredition). We use an STEncoder that shares the same parameters with that in STCRL to learn latent hidden features from the original data, where the learned data are then stored in the replay buffer. Finally, the learned features are input into a spatio-temporal decoder (STDecoder) for prediction.

4.2 Data Integration

To sample more representative samples from replay buffer $\mathcal{B}$, we design a novel ranking-based maximally interfered retrieval (RMIR) sampling method. Given the current data sequence D_i, we first sample M observations $\mathcal{X}_M$ =

$\left\langle X_{t_i^{k-M+1}}, \cdots X_{t_i^k} \right\rangle$ starting at time slot t_i^{k-M+1} from D_i and select samples $\mathcal{X}_{\mathcal{B}}$ stored in the replay buffer $\mathcal{B}$, where $\mathcal{B}$ is designed to act as an explicit memory to maintain a subset of previously learned observations (i.e., the previously trained observations without STMixup). Then, the selected samples are combined with the current observations through a proposed spatio-temporal mixup (STMixup) mechanism to alleviate catastrophic forgetting by benefiting from historical observations. We can formulate the process of data integration as follows.

$$\begin{aligned} \mathcal{X}_{\mathcal{B}} &= RMIR(\mathcal{B}, size = |\mathcal{S}|) \\ \mathcal{X}_{mix} &= STMixup(\mathcal{X}_M, \mathcal{X}_{\mathcal{B}}), \end{aligned} \tag{B.2}$$

where $|\mathcal{S}|$ is the sampling size.

We proceed to elaborate the *RMIR sampling method* and the *STMixup mechanism*.

RMIR Sampling Method

We design an RMIR sampling method to select $|\mathcal{S}|$ representative observations $\mathcal{X}_{\mathcal{B}}$ from the replay buffer. Typically, most methods in existing replay-based continuous learning methods randomly select observations from the replay memory, which leads to unsatisfied results about accuracy, due to the fact that they may ignore representative observations for replay. To select more representative samples, we first retrieve $|\mathcal{N}|$ observations $\mathcal{X}_{\mathcal{N}}$ which will be the most negatively impacted that suffer from an increase in loss by the update of foreseen parameters, where $|\mathcal{N}| > |\mathcal{S}|$. Specifically, given observation $X_{t_i^{\mathcal{B}}}$ in $\mathcal{B}$ and a standard objective function $\min_{\theta} \mathcal{L}_{RMIR}(f_\theta(X_{t_i^{\mathcal{B}}}), Y_{t_i^{\mathcal{B}}})$, where $f_\theta(\cdot)$ represents the URCL model, we update the parameters θ from $X_{t_i^{\mathcal{B}}}$ by gradient descent, as shown in Equation B.3.

$$\theta^v = \theta - \alpha \bigtriangledown \mathcal{L}_{RMIR}(f_\theta(X_{t_i^{\mathcal{B}}}), Y_{t_i^{\mathcal{B}}}), \tag{B.3}$$

where $\mathcal{L}_{RMIR}$ denotes sampling loss, $f_\theta(\cdot)$ represents the model, and $Y_{t_i^{\mathcal{B}}}$ is the ground truth. Note that Mean Absolute Error (MAE) is adopted as sampling loss. We select the top-$\mathcal{N}$ values.

Considering temporal correlations (e.g., trend and periodicity) of spatio-temporal data, data from a long time ago (periodic data) have a significant impact on the current prediction due to similarities. We then calculate the similarities between observations in $\mathcal{X}_{\mathcal{N}}$ and $\mathcal{X}_M$ by Pearson coefficient. Finally, we sample top-$\mathcal{S}$ observations of $\mathcal{X}_{\mathcal{N}}$ that are most similar to $\mathcal{X}_M$. In this way, we can not only select samples that can alleviate catastrophic forgetting, but also select samples to enhance temporal dependency capturing.

STMixup Mechanism

To benefit from historical observations, we introduce STMixup mechanism to fuse current observations $\mathcal{X}_M$ and observations $\mathcal{X}_{\mathcal{B}}$ sampled in $\mathcal{B}$. In particular, STMixup interpolates between $\mathcal{X}_M$ and $\mathcal{X}_{\mathcal{B}}$ to encourage the model to behave linearly across streaming spatio-temporal data sequences to minimize catastrophic forgetting.

Typically, assuming that (x_i, y_i) and (x_j, y_j) are two randomly selected feature-target pairs in the training data, STMixup generates virtual training examples by interpolation based on the principle of Vicinal Risk Minimization [47] to enlarge the support of the training distribution that overcomes concept drift. In STMixup, we use observation-groundtruth pairs to represent feature-target pairs in training. We use $(\tilde{x}, \tilde{y})$ to denote the interpolated feature-target pair in the vicinity of the raw two pairs.

$$\begin{aligned} \tilde{x} &= \lambda \cdot x_i + (1 - \lambda) \cdot x_j \\ \tilde{y} &= \lambda \cdot y_i + (1 - \lambda) \cdot y_j, \end{aligned} \tag{B.4}$$

where $\lambda \sim Beta(\alpha, \alpha)$, and $\alpha \in (0, \infty)$. We design STMixup by interpolating between current observations $\mathcal{X}_M$ and observations $\mathcal{X}_{\mathcal{B}}$ sampled from the replay buffer $\mathcal{B}$ by RMIR sampling. The interpolated observations $\mathcal{X}_{mix}$ after STMixup are formulated as follows:

$$\mathcal{X}_{mix} = \lambda \cdot \mathcal{X}_M + (1 - \lambda) \cdot \mathcal{X}_{\mathcal{B}} \tag{B.5}$$

The interpolated observations $\mathcal{X}_{mix}$ can enhance a model's ability to learn continuously by revisiting past instances in the replay buffer $\mathcal{B}$, that would be most negatively impacted by foreseen parameters. Moreover, STMixup can introduce an approximation of a regularized loss minimization [48] to avoid overfitting.

4.3 Spatio-temporal Continuous Representation Learning

The interpolated observations $\mathcal{X}_{mix}$ are input into the spatio-temporal continuous representation learning (STCRL) module for holistic feature learning. STCRL is a carefully designed self-supervised learning module, which consists of two parts: spatio-temporal data augmentation and an STSimsiam network. To enable effective spatio-temporal learning, we propose five customized spatio-temporal data augmentation methods by considering unique properties of spatio-temporal data, which transform a sample (i.e., observations in a sensor network) $\mathcal{G} = [\mathcal{X}_{mix}; G]$ to its corresponding perturbation $\mathcal{G}'$. We randomly select two different perturbations $\mathcal{G}'_1$ and $\mathcal{G}'_2$, and then input them into an STSimSiam network, which consists of two STEncoders $f_{\theta_{STE}}$ and a projection MLP head $h(\cdot)$. The aim of STSimSiam is to better capture spatio-temporal dependencies. Finally, STSimSiam maximizes the mutual infomation

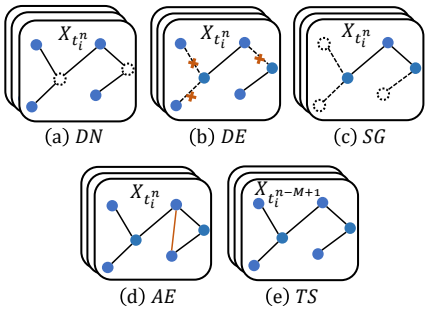


Fig. B.2: Spatio-temporal Data Augmentation

between the representations of two selected perturbations to ensure holistic feature preservation.

We proceed to elaborate the *spatio-temporal data augmentation* and the *STSim-Siam network*.

Spatio-Temporal Data Augmentation

We augment the interpolated observations $\mathcal{X}_{mix}$ using five carefully designed spatio-temporal augmentation methods, which builds semantically similar pairs and improves the quality of the learned representations by defeating perturbations. Although existing studies propose several data augmentation methods on graph data [49–51], they do not work well for spatio-temporal data due to complex spatial and temporal correlations, especially temporal correlations (e.g., closeness). Thus, we propose five spatially oriented data augmentation methods (i.e., DropNodes (DN), DropEdge (DE), SubGraph (SG), and AddEdge (AE)) and a temporally oriented method (i.e., TimeShifting (TS)). We cover each method next.

- **DN.** As shown in Figure B.2(a), given a sample $\mathcal{G} = [\mathcal{X}_{mix}; G]$, DN randomly discards a certain proportion (e.g., 10%) of the nodes in G to get $\mathcal{G}' = [\mathcal{X}_{mix}; G']$ following a definite distribution (e.g., a uniform distribution), which ensures that the missing nodes have no impact on the semantics (e.g., distribution) of G. In particular, we mask the entries in adjacency matrix A of that correspond to the discarded nodes to pertube the graph structure.

$$A'_{i,j} = \begin{cases} 0, & \text{if } v_i \text{ is discarded} \\ A_{i,j}, & \text{otherwise,} \end{cases} \tag{B.6}$$

  where $A'_{i,j}$ is an entry of the augmented matrix A'. Ideally, DN can promote model robustness by being less affected by missing data caused, e.g., sensor or communication failures.

- **DE.** DE randomly drops a part of edges, as shown in Figure B.2(b). As the weights of sensor network G are important to describe the spatial correlations between nodes (e.g., distance and similarity). We first sample a certain ratio of edges $\mathcal{E}$ from G following a specific distribution, and then set a threshold θ_{DE}. If the weights of edges in $\mathcal{E}$ are lower than θ_{DE}, we delete the corresponding edges, formulated as follows.

$$a_{i,j}^{'} = \begin{cases} 0, & \text{if } \mathrm{a_{i,j}} < \theta_{\mathrm{DE}} \\ a_{i,j}, & \text{otherwise,} \end{cases} \tag{B.7}$$

  where $a_{i,j}$ denotes the weight of the edge between nodes v_i and v_j, and $a'_{i,j}$ is the updated weight. The aim of the threshold is to retain important connectives of edges.

- **SG.** SG samples a subgraph $G' = (V', E')$ from $G = (V, E)$ by random walk to maximally preserve the semantics of a sensor network (cf. Figure B.2(c)), where $V' \subseteq V$ and $E' \subseteq E$. Through SG, we aim at improving local spatial correlations capturing by feature learning on the subgraph.

- **AE.** AE randomly selects a certain ratio of distant node pairs (e.g., more than three hops) and add edges between each node pair, which is shown in Figure B.2(d). The weights of these added edges are set as the dot product similarities of corresponding node pairs. Considering a node pair (v_i, v_j), the corresponding weight $w_{i,j}$ is calculated as follows.

$$w_{i,j} = \vec{\mathcal{X}}_{mix}^{i} \cdot \vec{\mathcal{X}}_{mix}^{j}, \tag{B.8}$$

  where $\vec{\mathcal{X}}_{mix}^{i}$ is a vector that represents features of node v_i. The aim of AE is to strengthen the power of our model to capture global spatial correlations by connecting distant node pairs that are similar to each other.

- **TS.** As shown in Figure B.2(e), TS, including time slicing, time warping, and time flipping, transforms current observations $\mathcal{X}_{mix}$ in $\mathcal{G}$ in the time domain. Note that we randomly select one of TS for model training.

  1. ***Time Slicing.*** Time slicing sub-samples the current observations $\mathcal{X}_{mix}$ in the time domain by randomly extracting continuous slice $\mathcal{X}_{mix}^{slice}$ with length l. Formally,

$$\mathcal{X}_{mix}^{slice} = \left\langle X_{mix,t_i^{s-l+1}}, \cdots, X_{mix,t_i^{s}} \right\rangle, \tag{B.9}$$

     where $t_i^{k-M+1} \leq t_i^{s-l+1} \leq t_i^{s} \leq t_i^{k}$, t_i^{k} is current time slot, M is the length of observations, and X_{mix,t_i^s} denotes observations after STMixup at time slot t_i^s

2. ***Time Warping.*** Time warping upsamples sliced observations $\mathcal{X}_{mix}^{slice}$ by linear interpolation to generate warped observations $\mathcal{X}_{mix}^{warp}$, the length of it is equal to that of $\mathcal{X}_{mix}$.

$$\mathcal{X}_{mix}^{warp} = \left\langle X'_{mix,t_i^{k-M+1}}, \cdots, X'_{mix,t_i^k} \right\rangle, \tag{B.10}$$

where X'_{mix,t_i^k} represents the generated observations at time slot t_i^k.

3. ***Time Flipping.*** Time flipping $TF(\cdot)$ flips the sign of warped observations $\mathcal{X}_{mix}^{warp}$ to generate a new sequence $\mathcal{X}_{mix}^{flip}$ in time domain as follows.

$$\mathcal{X}_{mix}^{flip} = \left\langle X'_{mix,t_i^k}, \cdots, X'_{mix,t_i^{k-M+1}} \right\rangle, \tag{B.11}$$

We randomly apply two different data augmentation methods to the integrated observations $\mathcal{X}_{mix}$ (generated by STMixup) to obtain two augmented observations $\mathcal{X}_{mix}^{aug1}$ and $\mathcal{X}_{mix}^{aug2}$.

STSimSiam Network

Inspired by exciting feature learning ability of self-supervised learning, we design a novel STSimSiam network under the guidance of self-supervised learning to capture holistic spatio-temporal representations, inputting two randomly augmented observations $\mathcal{X}_{mix}^{aug1}$ and $\mathcal{X}_{mix}^{aug2}$. More specifically, STSimSiam consists of two STEncoders to learn spatio-temporal representations of $\mathcal{X}_{mix}^{aug1}$ and $\mathcal{X}_{mix}^{aug2}$, respectively, and a projection head to project latent embeddings of $\mathcal{X}_{mix}^{aug1}$ into the latent space of $\mathcal{X}_{mix}^{aug2}$. Finally, considering that mutual information maximization is proven to learn holistic features and thus improves continuous learning [18], we maximize the mutual information between the representations learned from $\mathcal{X}_{mix}^{aug1}$ and $\mathcal{X}_{mix}^{aug2}$ by GraphCL loss.

As shown in the upper-right corner of Figure B.1, we input two randomly augmented observations $\mathcal{X}_{mix}^{aug1}$ and $\mathcal{X}_{mix}^{aug2}$ into the STSimSiam network. We first encode the two augmented observations as two fixed vectors z_1 and z_2 by STEncoders f_{θ_E} which are made of a specific spatio-temporal network, i.e., GraphWaveNet in this work, to capture the complex spatio-temporal dependencies. Next, we feed z_1 into a projection MLP head to map it into the latent space of z_2, which contains several MLP layers denoted as $h(\cdot)$. The process can be formulated as follows.

$$z_1 = f_{\theta_E}(\mathcal{X}_{mix}^{aug1}), \quad p_1 = h(z_1), \quad z_2 = f_{\theta_E}(\mathcal{X}_{mix}^{aug2}) \tag{B.12}$$

We use stopgrad operation $SG(\cdot)$ [52] to prevent the trivial solution obtained by STSimSiam. For example, $SG(z_2)$ denotes that the STEncoder on $\mathcal{X}_{mix}^{aug2}$ receives no gradient from z_2.

To enhance holistic feature preservation, we employ mutual information maximization to maximize the similarities between representations of $\mathcal{X}_{mix}^{aug1}$ and $\mathcal{X}_{mix}^{aug2}$. First, for the current two augmented observations, we use cosine similarity to measure similarities $C(\cdot)$ between their output vectors p_1 and z_2, formulated as follows.

$$\begin{aligned} C(p_1, z_2) &= \frac{p_1}{||p_1||_2} \cdot \frac{SG(z_2)}{||SG(z_2)||_2} \\ &= \frac{h(f_\theta \mathcal{X}_{mix}^{aug1})}{||h(f_\theta(\mathcal{X}_{mix}^{aug1})||_2} \cdot \frac{SG(f_\theta(\mathcal{X}_{mix}^{aug2}))}{||SG(f_\theta(\mathcal{X}_{mix}^{aug2}))||_2}, \end{aligned} \tag{B.13}$$

where $||\cdot||_2$ is l_2-$norm$.

We let $(\mathcal{X}_{mix}^{aug1}, \mathcal{X}_{mix}^{aug2})$ denote an augmented observation pair. For a minibatch of $\mathcal{S}$ augmented observation pairs, we adopt a GraphCL [49] loss to maximize their mutual information. The GraphCL loss L_{ssl}^s for the s-th augmented observation pair is defined as follows.

$$L_{ssl}^s = -log \frac{exp(C(p_{s,1}, z_{s,2})/\tau)}{\sum_{s'=1, s' \neq s}^{\mathcal{S}} exp(C(p_{s,1}, z_{s',2})/\tau)}, \tag{B.14}$$

where $p_{s,1}$ and $z_{s,2}$ denotes the output vectors of the s-th augmented observation pair $(\mathcal{X}_{mix,s}^{aug1}, \mathcal{X}_{mix,s}^{aug2})$, $C(\cdot)$ is cosine similarity, and τ is the temperature parameter. To extract more effective features [19], we define a symmetric similarity function and obtain final L_{ssl}^s.

$$L_{ssl}^s = -log \frac{exp((\frac{1}{2}C(p_{s,1}, z_{s,2}) + \frac{1}{2}C(p_{s,2}, z_{s,1}))/\tau)}{\sum_{s'=1, s' \neq s}^{\mathcal{S}} exp((\frac{1}{2}C((p_{s,1}, z_{s',2}) + \frac{1}{2}C(p_{s,2}, z_{s',1}))/\tau)}, \tag{B.15}$$

where $p_{s,2} = h(f_{\theta_E}(\mathcal{X}_{mix,s}^{aug2}))$ and $z_{s,1} = f_{\theta_E}(\mathcal{X}_{mix,s}^{aug1})$.

The final GraphCL loss is computed across all augmented pairs in the minibatch as follows.

$$L_{ssl} = \frac{1}{\mathcal{S}} \sum_{s=1}^{\mathcal{S}} L_{ssl}^s \tag{B.16}$$

where $\mathcal{S}$ is the batch size.

4.4 Spatio-Temporal Prediction

We design a spatio-temporal prediction network including a spatio-temporal encoder (STEncoder) and a spatio-temporal decoder (STDecoder). In particular, we input the interpolated observations $\mathcal{X}_{mix}$ into STEncoder $f_{\theta_E}(\cdot)$ to

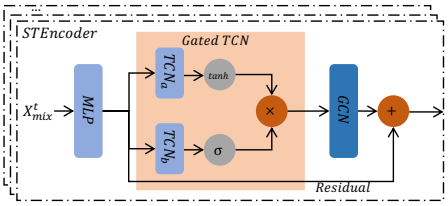


Fig. B.3: Illustation of STEncoder

learn high-dimensional representations through capturing complex spatio-temporal dependencies. The STEncoder shares parameters with the STEncoder in STSimSiam. The learned representations are input into STDecoder f_{θ_D} for prediction. Meanwhile, the recently learned data are stored in the replay buffer. The process can be formulated as follows.

$$h_\theta = f_{\theta_E}(\mathcal{X}_{mix}), \quad \hat{\mathcal{Y}} = f_{\theta_D}(h_\theta), \tag{B.17}$$

where $\hat{\mathcal{Y}}$ is the prediction.

One of the advantages of our framework is its generality. It can easily serve as a plug-in for most existing spatio-temporal prediction models that follow the autoencoder architecture. GraphWaveNet [9] is one of the state-of-the-art spatio-temporal prediction models to capture precise spatial dependencies and long-term temporal dependencies, but which is not developed under the autoencoder architecture. Inspired by its excellent performance on deep spatio-temporal graph modeling, we take it as an example of spatio-temporal prediction models in our work to show how to reorganize its architecture so as to conform to the autoencoder architecture (i.e., STEncoder and STDecoder). Specifically, in STEncoder, graph convolution layers integrated with gated temporal convolution layers are employed to capture spatio-temporal dependencies, as shown in Figure B.3, while several feed-forward networks are applied to map high-dimensional features into low-dimensional outputs for prediction in STDecoder, which is shown in Figure B.4. It is particularly noticeable that we study the effect of different spatio-temporal prediction models (including RNN-based DCRNN [4] and attention-based GeoMAN [53]) in our experimental part in Section 5.2.4. The studies show that our framework can adapt to different prediction models. For existing spatio-temporal prediction networks that lack an STDecoder, we employ stacked MLPs as the STDecoder.

STEncoder

The architecture of STEncoder is shown in Figure B.3. Taking $\mathcal{X}_{mix}$ as input, an MLP layer maps it into a high-dimensional latent space, and the learned features are then input into a Gated Temporal Convolution Network (TCN) layer to learn temporal correlations among observations of the input. Next, a Graph

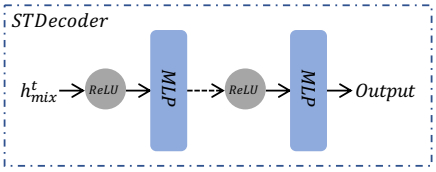


Fig. B.4: Illustation of STDecoder

Convolutional Network (GCN) layer is used to capture spatial correlations among the observations, where a residual operation is used to ensure accuracy. The learned features of the $i\text{-}th$ spatio-temporal layer can be formulated as follows.

$$h^i_\theta = f_\mathcal{G}(GatedTCN(W^i_\theta \cdot h^{i-1}_\theta + b^i_\theta), A), \tag{B.18}$$

where $f_\mathcal{G}(\cdot)$ denotes GCN, A is the adjacency matrix, W^i_θ is a learnable parameter, b^i_θ denotes the bias, and h^0_θ is euqal to $\mathcal{X}_{mix}$. The Gated TCN layer is composed of two parallel TCN layers (i.e., TCN_a and TCN_b).

Graph Convolution Layer. Recent studies pay considerable attention to generalize convolution networks for graph data. In this work, we use spectral convolutions on the constructed sensor network, which can be simply formulated as follows:

$$f_\mathcal{G}(X, A) = \sigma(\widetilde{A}XW^t), \tag{B.19}$$

where $f_\mathcal{G}$ represents the GCN operation, X denotes the node features, $\widetilde{A} = A + I_N$ is the adjacency matrix of $\mathcal{G}$ with added self-connections after normalizing, W^t denotes the learnable weight matrix, and $\sigma(\cdot)$ is the activation function.

Based on the first law of Geography: *"Near things are more related than distant things"* [1], we first construct a local spatial graph by considering the geographical distance between nodes. If two nodes v_i and v_j are connected with each other geographically, an edge between them exists, and the corresponding weight is set as follows:

$$A_{i,j} = \begin{cases} \frac{1}{dis}, & \text{if } v_i \text{ connects to } v_j \\ 0, & \text{otherwise,} \end{cases} \tag{B.20}$$

where dis denotes the geographical distance between two nodes v_i and v_j. Following Diffusion Convolutional Recurrent Neural Network (DCRNN) [4] that adopts diffusion GCN, we generalize the diffusion convolution layer into the form of Equation B.19 by modeling the diffusion process of graph signals with K finite steps, shown in Equation B.21.

$$f_\mathcal{G}(X, A) = \sigma(\sum_{k=0}^{K} P_k X W_k), \tag{B.21}$$

where P_k denotes the power series of the transition matrix. For an undirected graph, we can get $P = \widetilde{A}/rowsum(\widetilde{A})$, while for a directed graph, there are two directions of the diffusion process: forward direction $P^f = \widetilde{A}/rowsum(\widetilde{A})$ and backward direction $P^b = \widetilde{A}^T/rowsum(\widetilde{A}^T)$. The diffusion graph convolution layer for a directed graph is derived as:

$$f_{\mathcal{G}}(X, A) = \sigma(\sum_{k=0}^{K} P_k^f X W_{k_1} + P_k^b X W_{k_2}), \tag{B.22}$$

However, the first law of geography may not fully reflect the spatial correlation in urban areas, especially global spatial correlations (e.g., POI similarity) [2, 6]. To tackle this problem, we construct a self-adaptive adjacency matrix $\widetilde{A}_{adp}$ by multiplying two randomly initialized node embeddings with learnable parameters E_1 and E_2:

$$\widetilde{A}^{adp} = Softmax(ReLU(E_1 E_2^T)), \tag{B.23}$$

where *Softmax* is used to normalize the self-adaptive adjacency matrix. By comprehensively considering local and global spatial correlations, the final graph convolution layer is given in Equation B.24.

$$f_{\mathcal{G}}(X, A) = \sigma(\sum_{k=0}^{K} P_k^f X W_{k_1} + P_k^b X W_{k_2} + \widetilde{A}_k^{adp} X W_{k_3}), \tag{B.24}$$

If the sensor network is unknown, we adopt the self-adaptive adjacency matrix alone to capture the spatial dependencies.

Gated Convolution Layer. To capture temporal dependencies, we employ the dilated causal convolution [54] as our TCN due to its ability of modeling long-term temporal correlations and parallel computation. Specifically, given a data sequence $\mathcal{X}$ and a filter f, the dilated causal convolution operation of x at j-th step is represented as:

$$\mathcal{X} \circ f(j) = \sum_{m=0}^{K-1} f(m)\mathcal{X}(j - d \times m), \tag{B.25}$$

where d denotes the dilation factor that reflects skipping steps, and K is the length of filter f.

Gating mechanism is proven to be useful to control information flow through layers for TCN [55]. We adopt a simple form of Gated TCN in our work to model complex temporal correlations, where the Gated TCN only consists of an output gate. Given the observations $\mathcal{X}$, the Gated TCN is illustrated in Equation B.26.

$$h = g(W_1 \times \mathcal{X} + b) \odot \sigma(W_2 \times \mathcal{X} + c), \tag{B.26}$$

Algorithm 8 The URCL Framework

Input: Historical streaming spatio-temporal data sequences $\mathbb{D}$ from D_1 to D_m and a sensor network G
Output: $i\text{-}th$ URCL model θ_i

for D_i in $\mathbb{D}$ **do**
  $D_i^{train}, D_i^{val}, D_i^{test} \longrightarrow \emptyset$
  put training, validating and testing instances into D_i^{train}, D_i^{val} and D_i^{test}, respectively
  while $not\ converge$ **do**
    Sequentially select a batch of observations D_i^{batch} from D_i^{train}
    Sample previous observations $\mathcal{X}_{\mathcal{B}}$ from the replay buffer $\mathcal{B}$
    $\mathcal{X}_{mix} \longleftarrow$ data fusion with STMixup by Equation B.2
    $\mathcal{X}_{mix}^{aug1}, \mathcal{X}_{mix}^{aug2} \longleftarrow$ spatio-temporal data augmentation by DN, DE, SG, AE and TS
    $h_\theta \longleftarrow$ holistic feature learning by $STEncoder(\cdot)$ and GraphCL loss (i.e., Equation B.14)
    $\hat{\mathcal{Y}} \longleftarrow$ prediction with $STDecoder(\cdot)$ by Equation B.27
    Update θ_i based on Equation B.29
  end while
  Return the learned $i\text{-}th$ learned model θ_i
end for

where h is the learned features of input modeled by the first MLP layer, W_1 and W_2 are the learnable parameters, and b, c are the bias, $\odot$ denotes the element-wise product, $g(\cdot)$ and $\sigma(\cdot)$ represent activation functions (e.g., tanh and sigmod).

STDecoder

The learned features are then input into the STDecoder to decode the data representations for prediction. As shown in Figure B.4, the STDecoder contains several stacked feed-forward layers (i.e., MLPs) followed by activation functions (i.e., ReLU) to learn a projection function for future prediction. It can be formulated as follows.

$$\hat{\mathcal{Y}} = W_{\theta_D}(\alpha(h_\theta) + b_{\theta_D})), \tag{B.27}$$

where $\hat{\mathcal{Y}}$ denotes the prediction, W_{θ_D} is a learnable parameter, $\alpha(\cdot)$ is the ReLU activiation function, and b_{θ_D} is the bias.

4.5 Overall Objective Function

The overall objective of URCL is to minimize the prediction error by Mean Absolute Error (MAE) for each data stream D_i. The objective function is given

Table B.1: *Statistics of Datasets*

Dataset	***METR-LA***	***PEMS-BAY***	***PEMS04***	***PEMS08***
Area	Los Angeles	California	San Francisco Bay	San Bernaridino
Time span	4 months	5 months	2 months	2 months
Sampling interval	15 mins	15 mins	5 mins	5 mins
No. of Nodes	207	325	307	170
Input steps	12	12	12	12
Output steps	1	1	1	1

in the following:

$$L_{task} = \frac{1}{\mathcal{L}} \sum_{l=1}^{\mathcal{L}} |\hat{\mathcal{Y}}^l - \mathcal{Y}^l|, \tag{B.28}$$

where $\mathcal{L}$ is the training sample size, $\hat{\mathcal{Y}}^l$ is the prediction, and $\mathcal{Y}^l$ is the ground truth.

The final loss contains two parts: a prediction loss of the prediction task L_{task} and a GraphCL loss L_{ssl} for holistic feature learning. We combine them together and the overall loss is as follows.

$$L_{all} = L_{task} + L_{ssl} \tag{B.29}$$

The whole process of the URCL is shown in Algorithm 8, where lines 2–3 state the data preprocessing and lines 4–12 show the training process of URCL.

5 Experimental Evaluation

5.1 Experimental Setup

Datasets

The experiments are carried out on four widely-used graph-based urban spatio-temporal datasets: *METR-LA*, *PEMS-BAY*, *PEMS04* and *PEMS08*, where *METR-LA* and *PEMS-BAY* are related to traffic speed prediction, and *PEMS04* and *PEMS08* are related to traffic flow prediction.

- ***METR-LA.*** *METR-LA* is collected in Los Angeles County and contains the traffic data from March to June 2012 as collected by 207 sensors.
- ***PEMS-BAY.*** *PEMS-BAY* is collected in the Bay Area in California and contains data from 325 sensors. The time span is from January to May in 2017.
- ***PEMS04.*** *PEMS04* is collected from highways in the San Francisco Bay Area and contains 3848 detectors on 29 roads. The time span is from January to February in 2018.

- ***PEMS08.*** *PEMS08* is collectd in SanBernaridino from July to August in 2016. It contains the traffic data collected from 1,979 detectors on eight road segments.

We choose traffic prediction as a representative case of spatio-temporal prediction, as traffic prediction is a popular spatio-temporal prediction tasks. Dataset statistics are provided in Table B.1 and includes area, time span, sampling interval, number of nodes, and input and output steps.

Evaluation Methods.

We compare URCL with the following baseline methods.

- ***ARIMA.*** The Auto-Regressive Integrated Moving Average (ARIMA) method is a classic statistic-based method for time series prediction [8].
- ***DCRNN.*** The Diffusion Convolutional Recurrent Neural Network (DCRNN) method employs graph convolution networks to capture spatial correlations and the enocoder-decoder architecture with scheduled sampling to learn temporal correlations for traffic prediction [4].
- ***STGCN.*** The Spatio-Temporal Graph Convolutional Networks (STGCN) method applies ChebNet-GCN and 1D convolution to extract spatial and temporal dependencies for traffic prediction [56].
- ***MTGNN.*** The MTGNN model utilizes a graph-based deep learning method to exploit the inherent dependency relationships among multiple time series for multivariate time series forecasting [10].
- ***AGCRN.*** The Adaptive Graph Convolutional Recurrent Network (AGCRN) model captures the fine-grained spatial and temporal correlations automatically in traffic series data for traffic prediction [12].
- ***STGODE.*** The Spatial-Temporal Graph Ordinary Differential Equation Networks (STGODE) method adopts a tensor-based graph ordinary differential equation network to capture spatio-temporal dynamics and further is integrated with the temporal dilation convolution for traffic prediction [5].

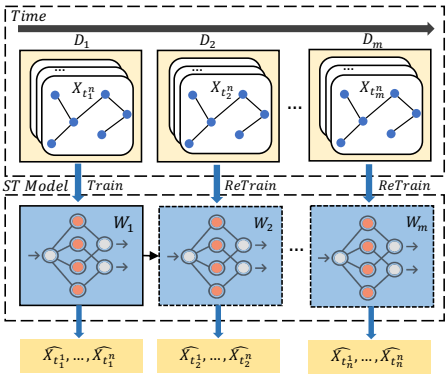


Fig. B.5: Illustration of Replay-based Streaming Spatio-Temporal Data Prediction

Metrics

Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) are adopted as the evaluation metrics, which are defined as follows.

$$\begin{aligned} MAE &= \frac{1}{M}\sum_{m=1}^{M}|\hat{\mathcal{Y}}^m - \mathcal{Y}^m| \\ RMSE &= \sqrt{\frac{1}{M}\sum_{m=1}^{M}||\hat{\mathcal{Y}}^m - \mathcal{Y}^m||^2}, \end{aligned} \tag{B.30}$$

where M is the testing data size, $\hat{y}^t$ is the prediction and y^t represents the ground truth. The smaller the AME and the RMSE are, the more accurate the method is. We also evaluate the efficiency of the models, including the training and inference (i.e., testing) time.

Other Implementation Details

We implement our model with the Pytorch framework on NVIDIA Quadro RTX 8000 GPU.

Continuous Learning Settings. We denote D_1 as a base set and D_2 to D_m as incremental sets, where $\mathbb{D} = \langle D_1, D_2, \cdots, D_m \rangle$ $(m \geq 1)$ is a sequence of streaming spatio-temporal data sequences as defined in Section 3. In the experiments, we use a base set, denoted as $\mathcal{B}_{set}$, and four incremental sets, denoted as $\mathcal{I}_{set}^1$ to $\mathcal{I}_{set}^4$, to facilitate continuous training. Specifically, we use 30% of each dataset as the base set and the remaining data in each dataset into four equal parts to form the incremental sets. The base set and incremental sets are provided sequentially with time evolving.

Training Process. We have tried to establish fair comparisons among the baseline methods (see Section 5.1) by mapping their original training process

Table B.2: *Performance of Training on Streaming Data on Two Datasets*

Method	PEMS-BAY										PEMS08									
	MAE					RMSE					MAE					RMSE				
	$\mathcal{B}_{set}$	$\mathcal{I}^1_{set}$	$\mathcal{I}^2_{set}$	$\mathcal{I}^3_{set}$	$\mathcal{I}^4_{set}$	$\mathcal{B}_{set}$	$\mathcal{I}^1_{set}$	$\mathcal{I}^2_{set}$	$\mathcal{I}^3_{set}$	$\mathcal{I}^4_{set}$	$\mathcal{B}_{set}$	$\mathcal{I}^1_{set}$	$\mathcal{I}^2_{set}$	$\mathcal{I}^3_{set}$	$\mathcal{I}^4_{set}$	$\mathcal{B}_{set}$	$\mathcal{I}^1_{set}$	$\mathcal{I}^2_{set}$	$\mathcal{I}^3_{set}$	$\mathcal{I}^4_{set}$
OneFitAll	1.21	2.76	2.72	3.75	3.63	2.17	5.82	4.88	7.28	6.86	18.15	38.75	39.32	44.22	27.21	26.28	44.29	57.34	63.01	37.19
FinetuneST	1.21	3.57	3.57	3.48	3.52	2.16	7.68	7.59	7.66	7.58	18.17	32.60	33.75	34.13	33.15	26.99	37.63	39.79	33.29	35.67
URCL	**1.12**	**1.09**	**1.21**	**1.19**	**1.15**	**1.91**	**1.86**	**2.08**	**2.01**	**1.99**	**18.13**	**18.07**	**17.45**	**17.38**	**17.20**	**26.44**	**26.02**	**25.65**	**24.86**	**24.52**

into a continuous training processes, as shown in Figure B.5. We first train an initial spatio-temporal (ST) model on base set D_1 and then re-train a new ST model on incremental sets (D_2–D_5) based on the last learned model.

Parameter Settings. The model parameters are set as follows. The input sizes for *METR-LA* and *PEMS-BAY* are 12×207×2 and 12×325×2, respectively, where 12 denotes the number of the previous time slots used for prediction, 307 and 325 are the numbers of the nodes, and 2 is the number of channels representing traffic speed and flow. Moreover, the input size for *PEMS04* and *PEMS08* are $12 \times 307 \times 3$ and $12 \times 170 \times 3$, respectively, where 3 denotes channels representing flow, speed, and occupancy. The STEncoder in URCL contains five layers, the hidden feature dimensions of which are 32, 32, 32, 32, and 256. The STDecoder contains two layers with 512 and 12 hidden features, respectively. The output of *METR-LA*, *PEMS-BAY*, *PEMS04*, and *PEMS08* are $1 \times 207 \times 1$, $1 \times 325 \times 1$, $1 \times 307 \times 1$, and $1 \times 170 \times 1$, respectively. We organize the buffer as a queue, whose size is set to 256. The parameters of the baseline methods are set based on their original papers and any accompanying code. Note that we normalize the streaming data into $[0, 1]$ to facilitate the feature learning.

5.2 Experimental Results

Performance of Training on Streaming Data

Our URCL proposes a replay-based strategy for training on streaming data, where a replay buffer is given to store previously learned samples that fuse with the training data by a spatio-temporal mixup mechanism to preserve historical knowledge effectively. To evaluate the performance of training on streaming data, two representative training strategies are used to replace the replay-based strategy in URCL and are compared with URCL: 1) ***OneFitAll*** that trains a model with the base set and then predicts all the testing data in the stream of spatio-temporal datasets; and 2) ***FinetuneST*** that trains an initial model on the base set and then fine-tunes the model with incremental sets repeatedly. GraphWaveNet is used as the base model of ***OneFitAll*** and ***FinetuneST***. We report the MAE and RMSE results on the PEMS-BAY and the PEMS08 datasets in Table B.2, where the overall best performance is marked in bold. To save space, we do not report results on the METR-LA and PEMS04 datasets, as these are similar to those obtained for PEMS-BAY and the PEMS08. The results on other datasets are included in the technical report[??].

We can see that URCL shows the best performance in terms of MAE and RMSE when compared with other methods on both datasets. More specifically, URCL outperforms the best among the baselines by 14.5%–67.3% and 15.5%–72.4% for MAE and RMSE on PEMS-BAY, respectively, and 1.2%–49.1% and 8.9%–35.5% for MAE and RMSE on PEMS08, respectively. OneFitAll and FinetuneST offer acceptable MAE and RMSE results on the base sets on both datasets, but their performance deteriorates on incremental sets. OneFitAll shows that as time goes, the new data can be different from the training data, showing that concept drift happens and a static model does not work, calling for a CL model. While a simple CL based FinetuneST is not enough, as it has forgetting problems. URCL achieves relatively stable performance on both base sets and incremental sets demonstrating its superiority.

Table B.3: Overall Accuracy on Four Datasets

Dataset		Metric	ARIMA	DCRNN	STGCN	MTGNN	AGCRN	STGODE	URCL
METR-LA	$\mathcal{B}_{set}$	MAE	4.99	4.01	3.92	4.01	3.83	7.39	**3.82**
		RMSE	8.82	6.22	7.98	6.89	6.33	11.42	**6.19**
	$\mathcal{I}^1_{set}$	MAE	5.87	10.55	9.99	5.83	6.78	7.11	**3.73**
		RMSE	9.77	14.41	14.77	9.71	12.42	11.62	**6.14**
	$\mathcal{I}^2_{set}$	MAE	5.88	7.59	3.83	4.33	4.30	7.35	**3.72**
		RMSE	10.49	8.44	5.63	6.89	7.59	11.77	**6.06**
	$\mathcal{I}^3_{set}$	MAE	4.88	4.77	4.96	3.74	4.32	6.45	**3.60**
		RMSE	10.49	6.81	7.00	5.91	7.59	10.17	**5.69**
	$\mathcal{I}^4_{set}$	MAE	5.00	4.13	3.89	4.31	4.32	7.40	**3.74**
		RMSE	10.99	6.25	**5.63**	6.94	7.53	11.78	6.01
PEMS-BAY	$\mathcal{B}_{set}$	MAE	1.98	1.30	1.31	1.39	1.25	4.89	**1.12**
		RMSE	3.85	2.24	2.28	2.53	2.51	7.93	**1.92**
	$\mathcal{I}^1_{set}$	MAE	3.88	1.11	2.57	1.30	1.14	4.66	**1.09**
		RMSE	4.98	1.87	5.58	2.22	2.23	7.61	**1.86**
	$\mathcal{I}^2_{set}$	MAE	3.67	1.30	2.20	1.47	1.74	2.44	**1.21**
		RMSE	4.89	2.34	4.55	2.53	2.24	3.69	**2.08**
	$\mathcal{I}^3_{set}$	MAE	3.24	1.21	3.33	1.20	1.62	5.31	**1.19**
		RMSE	6.77	2.11	7.06	2.13	2.01	8.64	**2.00**
	$\mathcal{I}^4_{set}$	MAE	5.92	1.22	3.49	1.17	1.73	4.84	**1.15**
		RMSE	10.22	2.00	7.53	2.10	2.05	7.84	**1.99**
PEMS04	$\mathcal{B}_{set}$	MAE	56.79	24.46	31.67	22.99	22.09	36.54	**19.77**
		RMSE	77.89	33.85	45.33	32.99	36.32	51.70	**31.26**
	$\mathcal{I}^1_{set}$	MAE	65.78	24.34	46.29	24.76	24.65	48.55	**21.56**
		RMSE	87.94	39.33	62.39	35.33	40.22	67.01	**34.10**
	$\mathcal{I}^2_{set}$	MAE	67.93	23.27	59.07	29.70	21.93	49.60	**21.61**
		RMSE	79.02	36.68	79.47	39.64	35.97	67.40	**34.04**
	$\mathcal{I}^3_{set}$	MAE	63.92	25.95	29.08	26.04	21.34	49.99	**20.97**
		RMSE	75.02	35.28	33.26	35.76	32.86	68.32	**32.12**
	$\mathcal{I}^4_{set}$	MAE	62.76	23.89	23.10	25.81	20.81	49.81	**20.48**
		RMSE	78.22	33.42	31.14	35.79	32.66	67.69	**32.20**
PEMS08	$\mathcal{B}_{set}$	MAE	45.98	23.07	18.33	20.70	23.03	28.58	**18.08**
		RMSE	67.28	32.24	27.68	30.38	38.44	38.54	**27.03**
	$\mathcal{I}^1_{set}$	MAE	55.22	20.35	18.35	20.66	19.47	36.55	**17.82**
		RMSE	75.33	27.92	**24.30**	29.82	31.74	46.04	26.37
	$\mathcal{I}^2_{set}$	MAE	49.33	22.15	39.32	20.27	17.45	36.06	**17.45**
		RMSE	59.89	29.63	57.34	28.97	27.35	45.25	**26.02**
	$\mathcal{I}^3_{set}$	MAE	58.99	19.94	44.22	22.68	17.70	36.12	**16.42**
		RMSE	65.79	27.04	63.01	30.24	27.56	45.04	**24.88**
	$\mathcal{I}^4_{set}$	MAE	62.89	19.87	27.21	18.92	18.35	36.05	**16.44**
		RMSE	71.36	27.91	37.19	26.99	28.80	45.30	**24.16**

Overall Accuracy

We report the MAE and RMSE values of the methods in Table B.3. The best performance by an existing method (ARIMA, DCRNN, STGCN, MTGNN,

AGCRN, and STGODE) is underlined, and the overall best performance is marked in bold. Note that we repeatably train each original baseline on each base and incremental to enable fair comparison with the replay-based strategy (like Figure B.5). The observations are in the following.

- The proposed URCL achieves the best results among all baselines in most cases, performing better than the best among the baselines by up to 36.0% and 34.1% in terms of MAE and RMSE, respectively. In most cases, URCL outperforms the best among the baselines in terms of MAE and RMSE on the traffic speed datasets (i.e., METR-LA and PEMS-BAY) except for $\mathcal{I}_{set}^{4}$ on METR-LA. Moreover, URCL also outperforms the best among the baselines in terms of MAE and RMSE on the traffic flow datasets (i.e., PEMS04 and PEMS08) except for $\mathcal{I}_{set}^{1}$ on PEMS08. We observe that the performance improvements obtained by URCL on the traffic speed datasets exceed those on the traffic flow datasets. This is because because much more traffic flow data is available than traffic speed data. Thus, the baselines using the replay-based training strategy with sufficient data can get comparable results.

- ARIMA, a traditional statistical method, performs worse than all methods except STGODE on METR-LA and PEMS-BAY and has the worst performance among all baselines on PEMS04 and PEMS08. This is because ARIMA only uses the time-series data of each region and ignores spatial dependencies. On METR-LA and PEMS-BAY, STGODE performs the worst since it is customized for speed prediction instead of flow prediction.

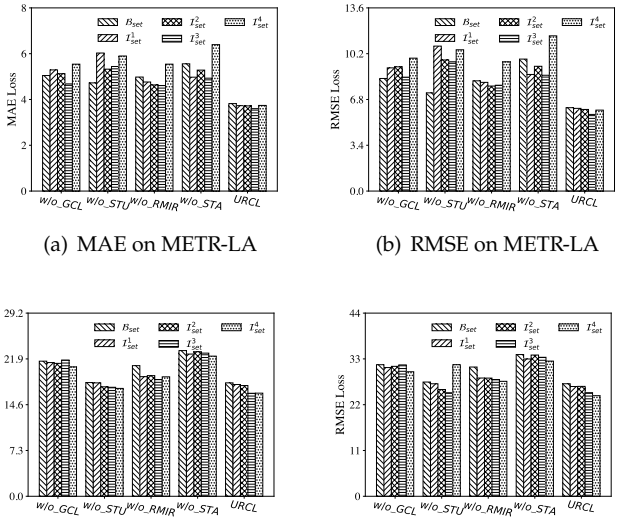
- As a popular spatio-temporal prediction model, AGCRN performs the best among all the baselines in most cases, due to its powerful ability to capture spatial and temporal correlations.

From the above observations, we can see that URCL can be applied on both traffic speed and traffic flow prediction tasks, which indicates generality across prediction tasks.

Ablation Study

To gain insight into the effects of key aspects of URCL, including the STMixup mechanism, the RMIR sampling strategy, the data augmentation methods, and the GraphCL loss, we evaluate four URCL variants:

- ***w/o STMixup (w/o_STU).*** *w/o_STU* is the URCL model without the STMixup module. We directly concatenate the original observations and sampled observations from the replay buffer.



(a) MAE on METR-LA (b) RMSE on METR-LA

(c) MAE on PEMS08 (d) RMSE on PEMS08

Fig. B.6: RMSE and MAE of URCL and Its Variants

- ***w/o RMIR Sampling (w/o_RMIR).*** *w/o RMIR* replaces the RMIR sampling mechanism from URCL with a random sampling one.
- ***w/o STAugmentation (w/o_STA).*** *w/o_STA* is the URCL model without the STAugmentation.
- ***w/o GraphCL (w/o_GCL).*** *w/o_GCL* is the URCL model without the GraphCL loss.

Figure B.6 shows the MAE and RMSE results on METR-LA and PEMS08. Regardless of the datasets, URCL always outperforms its counterparts without the STMixup mechanism, the RMIR sampling strategy, the data augmentation methods, or the GraphCL loss. These four components all improve the prediction accuracy of URCL, as removing any one of them increases the MAE and RMSE values on both base sets and incremental sets. Further, on both datasets, *w/o_STA* performs the worst among all variants in most cases. URCL outperforms *w/o_STA*, reducing the MAE and RMSE values by up to 41.5% and 47.8%, respectively, thus showing the benefit of spatio-temporal data augmentation. It is noteworthy that we include all the results in the technical report.

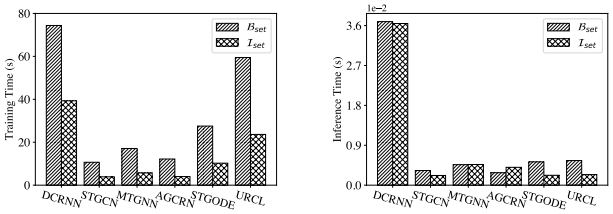
Effect of Different Backbones

Next, we study the effect of using different backbones, which are base models. For example, the backbone of URCL is the CNN-based GraphWaveNet [10] (see

Table B.4: Effect of Various Backbones on METR-LA and PEMS04

Dataset		Metric	DCRNN	GeoMAN	URCL
METR-LA	$\mathcal{B}_{set}$	MAE	4.97	5.11	3.82
		RMSE	8.18	8.47	6.19
	$\mathcal{I}^1_{set}$	MAE	4.76	4.87	3.73
		RMSE	8.07	8.28	6.14
	$\mathcal{I}^2_{set}$	MAE	4.64	4.41	3.72
		RMSE	7.80	7.09	6.06
	$\mathcal{I}^3_{set}$	MAE	4.61	4.39	3.60
		RMSE	7.86	7.28	5.69
	$\mathcal{I}^4_{set}$	MAE	5.54	5.27	3.74
		RMSE	9.59	8.77	6.01
PEMS04	$\mathcal{B}_{set}$	MAE	19.97	19.78	19.77
		RMSE	32.25	31.19	31.26
	$\mathcal{I}^1_{set}$	MAE	21.55	20.82	21.56
		RMSE	35.35	33.15	34.10
	$\mathcal{I}^2_{set}$	MAE	22.06	21.07	21.61
		RMSE	35.92	33.48	34.04
	$\mathcal{I}^3_{set}$	MAE	21.78	21.49	20.97
		RMSE	32.21	34.34	32.12
	$\mathcal{I}^4_{set}$	MAE	21.24	21.54	20.48
		RMSE	34.21	34.75	32.20

Section 4.4). In addition to CNN-based models, two main streams of existing graph-based spatio-temporal prediction models exist, including RNN-based and attention-based models that adopt RNNs and the attention mechanism to learn temporal dynamics, respectively, and employ graph neural networks to capture spatial correlations. We select two representative models as the backbones of URCL, i.e., DCRNN [4] that is based on RNNs. GeoMAN [53] that is based on the attention mechanism, We then compare them with URCL. For simplicity, we use the names of the backbones as the names of the compared methods. Table B.4 shows the prediction results on METR-LA and PEMS04. DCRNN and GeoMAN are neck-to-neck in terms of MAE and RMSE on both datasets. URCL has the best performance in most cases, but the performance of the other two models are comparable especially on PEMS04, demonstrating the generality of URCL for adopting different backbones.



(a) Training Time (b) Inference Time

Fig. B.7: Training and Inference Time on PEMS04

Efficiency

As efficiency is important for spatio-temporal prediction on streaming data, we study the training time (of each epoch) and inference (testing) time (of each

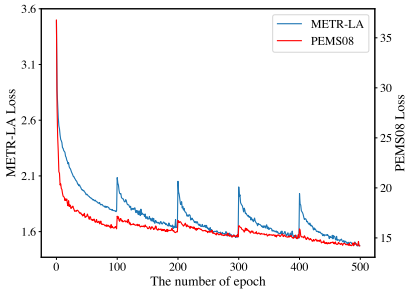


Fig. B.8: Training Convergence of URCL on MATR-LA and PEMS08

observation) for all the deep-learning-based methods on PEMS04. Figure B.7 shows the training and inference time of the base sets, as well as the averages of those of all the incremental sets. One can see that the training time of URCL is lower than that of DCRNN, as shown in Figure B.7(a). Although the baselines (except DCRNN) take less time for training, they perform worse than URCL in terms of prediction accuracy. Figure B.7(b) shows that the inference time of URCL is far lower than that of DCRNN and is comparable with those of the other baselines, which indicates the feasibility of URCL for model deployment in real streaming spatio-temporal prediction scenarios.

Convergence Analysis of Training

We study the training convergence of URCL on MATR-LA and PEMS08—see Figure B.8. The number of epochs for training each base/incremental set is 100, which means that we use the first 100 epochs for training $\mathcal{B}_{set}$, the second 100 epochs for training $\mathcal{I}^1_{set}$, the third 100 epochs for training $\mathcal{I}^2_{set}$, and the rest can be done in the same manner. We observe that URCL converges after approximately 90 epochs on $\mathcal{B}_{set}$ of both datasets, which shows that it converges quickly. Moreover, URCL converges after around 60 epochs on all incremental sets of both datasets, meaning that it converges faster than on the base set. This shows that URCL can reduce the training time on the unseen data substantially and thus can reduce computational costs. In addition, the training loss curves drop smoothly, after which there are minor fluctuations on the training loss. This is mainly because the proposed STMixUp mechanism brings the benefits of regularization.

6 Conclusion

We propose a unified replay-based continuous learning framework for spatio-temporal prediction on streaming data, that aims to capture the complex patterns in spatio-temporal data. To embrace historical knowledge, an STMixup

mechanism is designed to integrate training data with representative data sampled from a replay buffer. By coupling a spatio-temporal SimSiam network with a spatio-temporal prediction autoencoder, the holistic features are extracted through mutual information maximization. In addition, five spatio-temporal data augmentation methods are proposed to cope with the SimSiam network. An empirical study with real datasets offers evidence that the paper's proposals improve on the state of the art in terms of prediction accuracy. An interesting research direction is to attempt to further improve the training efficiency of the proposed URCL.

References

[1] J. Zhang, Y. Zheng, and D. Qi, "Deep spatio-temporal residual networks for citywide crowd flows prediction," in *AAAI*, 2017, pp. 1655–1661.

[2] H. Miao, J. Shen, J. Cao, J. Xia, and S. Wang, "MBA-STNet: Bayes-enhanced discriminative multi-task learning for flow prediction," *TKDE*, vol. 14, no. 8, pp. 1–14, 2022.

[3] J. Ji, J. Wang, Z. Jiang, J. Jiang, and H. Zhang, "STDEN: Towards physics-guided neural networks for traffic flow prediction," in *AAAI*, vol. 36, no. 4, 2022, pp. 4048–4056.

[4] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," in *ICLR*, 2018, pp. 1–10.

[5] Z. Fang, Q. Long, G. Song, and K. Xie, "Spatial-temporal graph ode networks for traffic flow forecasting," in *SIGKDD*, 2021, pp. 364–373.

[6] H. Yao, X. Tang, H. Wei, G. Zheng, and Z. Li, "Revisiting spatial-temporal similarity: A deep learning framework for traffic prediction," in *AAAI*, 2019, pp. 5668–5675.

[7] B. L. Smith and M. J. Demetsky, "Traffic flow forecasting: comparison of modeling approaches," *Journal of transportation engineering*, vol. 123, no. 4, pp. 261–266, 1997.

[8] S. Shekhar and B. M. Williams, "Adaptive seasonal time series models for forecasting short-term traffic flow," *TRR*, vol. 2024, no. 1, pp. 116–125, 2007.

[9] Z. Wu, S. Pan, G. Long, J. Jiang, and C. Zhang, "Graph wavenet for deep spatial-temporal graph modeling," in *IJCAI*, 2019, pp. 1907–1913.

[10] Z. Wu, S. Pan, G. Long, J. Jiang, X. Chang, and C. Zhang, "Connecting the dots: Multivariate time series forecasting with graph neural networks," in *SIGKDD*, 2020, pp. 753–763.

[11] R.-G. Cirstea, T. Kieu, C. Guo, B. Yang, and S. J. Pan, "EnhanceNet: Plugin neural networks for enhancing correlated time series forecasting." in *ICDE*, 2021, pp. 1739–1750.

[12] L. Bai, L. Yao, C. Li, X. Wang, and C. Wang, "Adaptive graph convolutional recurrent network for traffic forecasting," in *NeurIPS*, 2020, pp. 17 804–17 815.

[13] J. Serra, D. Suris, M. Miron, and A. Karatzoglou, "Overcoming catastrophic forgetting with hard attention to the task," in *ICML*, 2018, pp. 4548–4557.

[14] J. Hofmanninger, M. Perkonigg, J. A. Brink, O. Pianykh, C. Herold, and G. Langs, "Dynamic memory to alleviate catastrophic forgetting in continuous learning settings," in *MICCAI*, 2020, pp. 359–368.

[15] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[16] R. Aljundi, E. Belilovsky, T. Tuytelaars, L. Charlin, M. Caccia, M. Lin, and L. Page-Caccia, "Online continual learning with maximal interfered retrieval," in *NeurIPS*, 2019, pp. 11 849–11 860.

[17] F.-K. Sun, C.-H. Ho, and H.-Y. Lee, "Lamol: Language modeling for lifelong language learning," *ICLR*, 2019.

[18] Y. Guo, B. Liu, and D. Zhao, "Online continual learning through mutual information maximization," in *ICML*, 2022, pp. 8109–8126.

[19] X. Chen and K. He, "Exploring simple siamese representation learning," in *CVF*, 2021, pp. 15 750–15 758.

[20] J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny, "Barlow twins: Self-supervised learning via redundancy reduction," in *ICML*, 2021, pp. 12 310–12 320.

[21] S. Wang, J. Cao, and P. S. Yu, "Deep learning for spatio-temporal data mining: A survey," *TKDE*, vol. 34, no. 8, pp. 3681–3700, 2020.

[22] J. Ji, J. Wang, C. Huang, J. Wu, B. Xu, Z. Wu, J. Zhang, and Y. Zheng, "Spatio-temporal self-supervised learning for traffic flow prediction," in *AAAI*, vol. 37, no. 4, 2023, pp. 4356–4364.

[23] Z. Zhou, K. Yang, Y. Liang, B. Wang, H. Chen, and Y. Wang, "Predicting collective human mobility via countering spatiotemporal heterogeneity," *TMC*, pp. 1–16, 2023.

[24] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," in *NeurIPS*, 2015, pp. 802–810.

[25] X. Yi, J. Zhang, Z. Wang, T. Li, and Y. Zheng, "Deep distributed fusion network for air quality prediction," in *SIGKDD*, 2018, pp. 965–973.

[26] M. Lippi, M. Bertini, and P. Frasconi, "Short-term traffic flow forecasting: An experimental comparison of time-series analysis and supervised learning," *TITS*, vol. 14, no. 2, pp. 871–882, 2013.

[27] T. Kieu, B. Yang, C. Guo, R.-G. Cirstea, Y. Zhao, Y. Song, and C. S. Jensen, "Anomaly detection in time series with robust variational quasi-recurrent autoencoders," in *ICDE*, 2022, pp. 1342–1354.

[28] Y. Zhao, X. Chen, L. Deng, T. Kieu, C. Guo, B. Yang, K. Zheng, and C. S. Jensen, "Outlier detection for streaming task assignment in crowdsourcing," in *WWW*, 2022, pp. 1933–1943.

[29] T. Kieu, B. Yang, C. Guo, C. S. Jensen, Y. Zhao, F. Huang, and K. Zheng, "Robust and explainable autoencoders for time series outlier detection," in *ICDE*, 2022, pp. 3038–3050.

[30] S. Mo, Z. Bao, B. Zheng, and Z. Peng, "Towards an optimal bus frequency scheduling: When the waiting time matters," *TKDE*, vol. 34, no. 9, pp. 4484–4498, 2020.

[31] X. Chen, L. Deng, F. Huang, C. Zhang, Z. Zhang, Y. Zhao, and K. Zheng, "Daemon: Unsupervised anomaly detection and interpretation for multivariate time series," in *ICDE*, 2021, pp. 2225–2230.

[32] X. Chen, L. Deng, Y. Zhao, and K. Zheng, "Adversarial autoencoder for unsupervised time series anomaly detection and interpretation," in *WSDM*, 2023, pp. 267–275.

[33] D. Campos, M. Zhang, B. Yang, T. Kieu, C. Guo, and C. S. Jensen, "LightTS: Lightweight time series classification with adaptive ensemble distillation," *SIGMOD*, vol. 1, no. 2, pp. 171:1–171:27, 2023.

[34] X. Wu, D. Zhang, M. Zhang, C. Guo, B. Yang, and C. S. Jensen, "AutoCTS+: Joint neural architecture and hyperparameter search for correlated time series forecasting," *SIGMOD*, vol. 1, no. 1, pp. 97:1–97:26, 2023.

[35] X. Wu, D. Zhang, C. Guo, C. He, B. Yang, and C. S. Jensen, "AutoCTS: Automated correlated time series forecasting," *PVLDB*, vol. 15, no. 4, pp. 971–983, 2022.

[36] M. Li and Z. Zhu, "Spatial-temporal fusion graph neural networks for traffic flow forecasting," in *AAAI*, 2021, pp. 4189–4196.

[37] Z. Zhou, Q. Huang, K. Yang, K. Wang, X. Wang, Y. Zhang, Y. Liang, and Y. Wang, "Maintaining the status quo: Capturing invariant relations for OOD spatiotemporal learning," in *SIGKDD*, 2023, pp. 3603–3614.

[38] Y. Cheng, P. Chen, C. Guo, K. Zhao, Q. Wen, B. Yang, and C. S. Jensen, "Weakly guided adaptation for robust time series forecasting," *PVLDB*, 2024.

[39] K. Zhao, C. Guo, P. Han, M. Zhang, Y. Cheng, and B. Yang, "Multiple time series forecasting with dynamic graph modeling," *PVLDB*, 2024.

[40] Z. Chen and B. Liu, "Lifelong machine learning," *Synthesis lectures on artificial intelligence and machine learning*, vol. 12, no. 3, pp. 1–207, 2018.

[41] S. Thrun, "Is learning the n-th thing any easier than learning the first?" in *NeurIPS*, 1995, pp. 640–646.

[42] P. Ruvolo and E. Eaton, "Ella: An efficient lifelong learning algorithm," in *ICML*, 2013, pp. 507–515.

[43] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars, "Memory aware synapses: Learning what (not) to forget," in *ECCV*, 2018, pp. 139–154.

[44] X. Chen, J. Wang, and K. Xie, "TrafficStream: A streaming traffic flow forecasting framework based on graph neural networks and continual learning," in *IJCAI*, 2021, pp. 3620–3626.

[45] Y. Xiao, M. Liu, Z. Zhang, L. Jiang, M. Yin, and J. Wang, "Streaming traffic flow prediction based on continuous reinforcement learning," in *ICDMW*, 2022, pp. 16–22.

[46] M. Tubaishat and S. Madria, "Sensor networks: an overview," *Potentials*, vol. 22, no. 2, pp. 20–23, 2003.

[47] O. Chapelle, J. Weston, L. Bottou, and V. Vapnik, "Vicinal risk minimization," in *NeurIPS*, 2000, pp. 1–7.

[48] L. Zhang, Z. Deng, K. Kawaguchi, A. Ghorbani, and J. Zou, "How does mixup help with robustness and generalization?" *ICLR*, 2020.

[49] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, "Graph contrastive learning with augmentations," in *NeurIPS*, 2020, pp. 5812–5823.

[50] S. Thakoor, C. Tallec, M. G. Azar, M. Azabou, E. L. Dyer, R. Munos, P. Veličković, and M. Valko, "Large-scale representation learning on graphs via bootstrapping," *ICLR*, 2021.

[51] J. Zeng and P. Xie, "Contrastive self-supervised learning for graph classification," in *AAAI*, 2021, pp. 10 824–10 832.

[52] D. Madaan, J. Yoon, Y. Li, Y. Liu, and S. J. Hwang, "Representational continuity for unsupervised continual learning," in *ICLR*, 2021, pp. 1–18.

[53] Y. Liang, S. Ke, J. Zhang, X. Yi, and Y. Zheng, "GeoMAN: Multi-level attention networks for geo-sensory time series prediction," in *IJCAI*, 2018, pp. 3428–3434.

[54] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *ICLR*, 2016. [Online]. Available: http://arxiv.org/abs/1511.07122

[55] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, "Language modeling with gated convolutional networks," in *ICML*, 2017, pp. 933–941.

[56] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting," in *IJCAI*, 2018, pp. 3634–3640.

Paper C

LightTR+: A Lightweight Incremental Framework for Federated Trajectory Recovery

Hao Miao, Ziqiao Liu, Yan Zhao, Chenxi Liu, Chenjuan Guo, Bin Yang, Kai Zheng, Huan Li, Christian S. Jensen

The paper has been submitted to the
IEEE Transactions on Knowledge and Data Engineering, 2024.

© 201X IEEE
The layout has been revised.

With the proliferation of GPS-equipped edge devices, huge trajectory data are generated and accumulated in various domains, driving numerous urban applications. However, due to the limited data acquisition capabilities of edge devices, many trajectories are often recorded at low sampling rates, reducing the effectiveness of these applications. To address this issue, we aim to recover high-sample-rate trajectories from low-sample-rate ones enhancing the usability of trajectory data. Recent approaches to trajectory recovery often assume centralized data storage, which can lead to catastrophic forgetting, where previously learned knowledge is entirely forgotten when new data arrives. This not only poses privacy risks but also degrades performance in decentralized settings where data streams into the system incrementally. To enable decentralized training and streaming trajectory recovery, we propose a Lightweight incremental framework for federated Trajectory Recovery, called LightTR+, which is based on a client-server architecture. Given the limited processing capabilities of edge devices, LightTR+ includes a lightweight local trajectory embedding module that enhances computational efficiency without compromising feature extraction capabilities. To mitigate catastrophic forgetting, we propose an intra-domain knowledge distillation module. Additionally, LightTR+ features a meta-knowledge enhanced local-global training scheme, which reduces communication costs between the server and clients, further improving efficiency. Extensive experiments offer insight into the effectiveness and efficiency of LightTR+.

1 Introduction

Thanks to the rapid adoption and advancement of mobile sensing devices, massive amounts of trajectory data are being collected in a decentralized manner, enabling a wide range of trajectory-based applications [1–3], such as traffic prediction [4], destination prediction [5], and vehicle navigation [1]. However, in practice, trajectory data is often sampled at a low rate [6], resulting in incomplete trajectories (a.k.a. low-sampling-rate trajectories), which reduces the effectiveness of the above-mentioned applications due to the loss of detailed information and increased uncertainty. Thus, it is important to recover missing points in incomplete trajectories, a process known as *trajectory recovery*, to enable more effective utilization of low-sampling-rate trajectories.

Due to its significance, considerable research efforts have been dedicated to designing effective trajectory recovery models [7]. Traditionally, statistical models were employed to recover incomplete trajectories using historical trajectory data [8, 9], often in conjunction with various map-matching algorithms that align GPS coordinates with corresponding road segments. More recent studies [10–12] apply neural networks to recover trajectories by learning deep representations of the trajectories. These networks typically consist of a stack of

spatio-temporal (ST) blocks designed to capture the complex spatio-temporal dependencies of trajectories. An ST-block contains base *ST-operators*, which can be categorized into convolutional neural networks (CNN) [13], recurrent neural networks (RNN) [11, 14], or attention neural networks (Attn) [10] based *ST-operators*. However, existing methods assume that models are trained with centralized data collected from edge devices, leading to high collection and storage costs and failing to handle decentralized training data. Additionally, with increasing privacy concerns, individuals may be unwilling to disclose their raw trajectories to untrusted data providers, worrying that their data can be misused by malicious third parties. Thus, decentralized trajectory processing, which enables privacy protection, has become a key issue in current trajectory-based applications to comply with licensing agreements and data access restrictions [4].

As a result, we need a decentralized learning model that can adapt to decentralized trajectory data and learn complex spatio-temporal correlations effectively. Recently, federated learning (FL) has emerged as a solution for training models with decentralized data spread across multiple clients (e.g., mobile devices, organizations, or platform centers) while ensuring privacy. In FL, numerous clients train a model under the coordination of a central server, keeping the training data decentralized. However, existing FL-based methods [15, 16] do not consider inherent spatio-temporal dependencies, which are crucial for effective trajectory embedding [11]. In this study, we aim to develop a novel FL-based trajectory recovery model that combines decentralized data processing and complex spatio-temporal dependency modeling. Developing such a model is a non-trivial task due to the following four challenges.

Challenge I: scalability. Existing popular trajectory recovery methods often suffer from poor scalability, as deep learning-based models are often large, rendering both training and inference time-consuming and computationally expensive. This limitation is particularly problematic for resource-constrained edge computing devices, which are essential for decentralized computation. In addition, existing methods may incur memory overflow in large-scale trajectory learning settings since the entire network must reside in memory during training. For example, given N trajectories, each with L points, the memory cost of attention-based spatio-temporal operators (*ST-operators*) increases quadratically with L and N (see Table C.1). Currently, no customized lightening module exists in trajectory recovery models, and simply lightening existing models degrades their performance dramatically [17], further limiting the scalability of trajectory learning.

Challenge II: communication cost. During training in FL, multiple rounds of communication occur between the central server and all participating clients. Limited network bandwidth and a large number of participating clients can create a communication bottleneck, leading to increased latencies and reduced practicality. Statistically, the collected trajectories across different clients

are usually not independent and identically distributed (Non-IID). This non-uniformity (or heterogeneity) leads to a significant increase in the communication rounds needed to achieve convergence, complicating the development of an optimal global model. The communication capacities of clients may differ due to hardware constraints, network connections, and available power. Thus, it is highly desirable, yet challenging, to develop a communication cost reduction method in federated trajectory recovery that can address these issues.

Challenge III: Domain Interference. Mitigating both inter- and intra-domain interference in FL for trajectory recovery is particularly challenging. FL typically involves two phases: aggregation and local updating [15, 18]. During the aggregation phase, the central server collects and combines local parameters from active clients. This process can introduce potential inter-interference by other clients, where the global model becomes confused due to the heterogeneous (dissimilar) data from various clients. This inter-interference can hinder model convergence and degrade performance. Thus, we need to learn effective meta-knowledge across clients to minimize inter-domain interference (i.e., heterogeneities) across clients and maximize inter-client knowledge transfer.

In the local updating phase, each client optimizes its local model with continuously-generated private data, which can lead to overfitting to the current knowledge and forgetting of previous knowledge, called intra-domain interference, which results in catastrophic forgetting [19]. While many efforts have been made to use incremental learning (IL) techniques to address intra-domain interference in computer vision [18, 19], most such techniques are centralized and do not consider privacy issues, making them unsuitable for decentralized data. In addition, no IL-based methods exist that are effective at capturing the unique spatio-temporal correlations unique to trajectories.

Challenge IV: Streaming learning. Trajectory recovery often involves dealing with data arrives in real-time from various sources such as GPS devices and mobile applications. This makes it inherently a streaming challenge due to the continuous data generation, requiring models to adapt in real time. This continuous setting is challenging for the traditional FL models, which typically assume static datasets. These models are trained once on a particular dataset and then applied to another dataset, with the underlying assumption that both datasets follow the same distribution. However, in real-world scenarios, trajectory data is generated incrementally, and concept drift—where data distributions change over time—occurs frequently in streaming trajectories. As a result, directly applying these models to streaming trajectories can deteriorate model performance considerably [19, 20]. This yields a new challenge: developing an FL model that can incrementally adapt to the incoming data and consistently perform trajectory recovery tasks over time.

In the preliminary version of this study [21], we developed a LightTR framework to solve *Challenges I* and *II*. Specifically, to avoid high memory consumption and limited scalability (*Challenge I*), we designed a local lightweight

trajectory embedding (LTE) model for each client. The LTE model contains an embedding component and a stack of ST-blocks to learn effective spatio-temporal representations. Unlike previous studies [14, 22], we formalized a lightweight *ST-operator* in the ST-blocks and replaced the popular *ST-operators* (e.g., CNNs and Attns) with a pure MLP (multi-layer perceptron) architecture, considering its lower space complexity (i.e., $O(L+D+1)$) and time complexity (i.e., $O(N \cdot (L+D))$), where L denotes the number of points in a trajectory, D is the embedding size, and N is the number of trajectories. We use only one RNN layer combined with an MLP to ensure the capture of temporal dependencies. To reduce communication costs and accelarate model convergence (*Challenge II*), we proposed a meta-knowledge enhanced local-global training module based on knowledge distillation. Before federated training, we introduced a Teacher model (i.e., a meta-learner) to learn local meta-knowledge for each client using a portion of its local data. The local lightweight trajectory embedding model acts as the student model. During FL, the Teacher model guides the optimization of student models to learn better common features and achieve faster convergence.

However, LightTR [21] is not equipped to tackle *Challenges III* and *IV*. Therefore, we propose LightTR+, a lightweight incremental framework for federated trajectory recovery based on horizontal federated learning (i.e., a server-clients architecture). The objective is to collaboratively train models by maintaining a shared global model on a central server while utilizing the data of each client in a decentralized fashion. Specifically, to alleviate intra-domain interference (*Challenge III*), we propose an intra-domain knowledge distillation module. This module leverages the previously trained local trajectory embedding model to transfer knowledge of old data to the current data for each client in the local updating phase. Additionally, the proposed meta-knowledge enhanced local-global training module can learn effective meta-knowledge to alleviate data heterogeneity across clients, thereby limiting inter-domain interference (*Challenge III*) in the aggregation phase. To address *Challenge IV*, we save the local trajectory model learned on previous local data and use it to constrain subsequent local updating with a distillation loss. When new data arrives, we distill the knowledge from the saved model to the newly updated model learned on the current data. Our goal is to enable the local model on each client to perform well on incoming data while preserving the knowledge learned from previous data as much as possible.

The major value-added extensions over our previous study [21] are summarized as follows.

- We provide an in-depth analysis of the two key limitations in our previous federated trajectory recovery framework: its inability to contend with domain interference and to handle streaming trajectories effectively.
- To the best of our knowledge, this is the first study to systematically

explore federated incremental learning on decentralized trajectory data, ensuring privacy protection for locally collected trajectory data. We propose a lightweight federated incremental framework entitled LighTR+ for trajectory recovery, which leverages knowledge distillation to alleviate catastrophic forgetting.

- We report on experiments using real data, providing insight into the effectiveness and efficiency of the proposed LightTR+ framework in alleviating catastrophic forgetting on decentralized trajectories.

The remainder of this paper is organized as follows. Section 2 covers preliminary concepts and formalizes the problem of federated trajectory recovery. Section 3 analyzes the drawbacks of existing trajectory learning methods. We report the design of the LightTR+ framework in Section 4, followed by the experimental study in Section 5. Section 6 covers related work, and Section 7 concludes the paper.

2 Preliminaries

We proceed to present necessary preliminaries and then define the problem addressed.

Definition 25 (Road Network) *A road network is a directed graph $G = (V, E)$, where V is a road vertex set, and E is a road edge set. Each $v_i \in V$ denotes a road segment intersection or a road end, and each $e_{i,j} \in E$ denotes a directed road segment from v_i to v_j.*

Definition 26 (GPS Point) *A GPS point can be defined as a triple $p = \langle lat, lng, \gamma \rangle$, which captures the latitude lat and longitude lng of the GPS position including additional information γ, e.g., address.*

Definition 27 (Raw Incomplete Trajectory) *A raw incomplete trajectory τ is a sequence of n points, i.e., $\tau = \langle (p_1, t_1), (p_2, t_2), \cdots, (p_n, t_n) \rangle$, where t_i denotes the timestamp of p_i.*

Definition 28 (Sampling Rate) *A sampling rate ϵ is the time difference between two consecutively sampled points, which is usually determined by the sampling device.*

In practice, the sampling rate often changes near ϵ in most cases because of the inherent temporal bias of trajectory data. It is also worth noting that the collected trajectory points may not be precise due to GPS device measurement errors and GPS noises. Map matching is usually adopted to snap raw points onto the road network [11, 23–25]. In this study, we use a map-matching method based on deep hierarchical networks (DHN) [26] to convert a sequence of raw latitude/longitude coordinates for map matching.

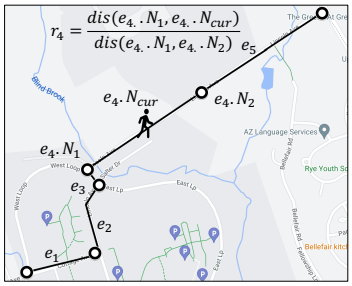


Fig. C.1: Moving Ratio

Definition 29 (Map-matched ϵ-Sampling-Rate Trajectory) *A map-matched trajectory T with ϵ-sampling rate is a sequence of map-matched trajectory points, i.e.,*

$$T = \langle (\widetilde{p}_1, t_1), \cdots, (\widetilde{p}_i, t_i), \cdots, (\widetilde{p}_n, t_n) \rangle, \tag{C.1}$$

where $t_{i+1} - t_i = \epsilon$, $\widetilde{p}_i = \langle e, r \rangle$ is the map-matched (trajectory) point at timestamp t_i, e denotes the road segment, and r is the moving ratio. Next, r represents the ratio between the current moving distance (from the start node of e to the current node) and the length of road segment e, i.e., $r = \frac{dis(e.N_1, e.N_{cur})}{dis(e.N_1, e.N_2)}$, where $dis(a, b)$ denotes the distance between nodes a and b, $e.N_1$ denotes the start node of e, $e.N_{cur}$ denotes the current node of e, and $e.N_2$ denotes the end node of e.

As shown in Figure 2, each road segment e covers the start node $e.N_1$ and end node $e.N_2$, and records the distance between the two points. Next, the moving ratio r will be determined by the moving distance (from $e.N_1$ to the current node $e.N_{cur}$) and the total length of e. For example, if the current node is located in the middle of e_4, we can obtain that $r = 0.5$.

Definition 30 (Incomplete Map-matched ϵ-Sampling-Rate Trajectory) *An incomplete map-matched ϵ-sampling-rate trajectory, i.e.,*

$$T_{icp} = \langle (\widetilde{p}_1, t_1), \cdots, (\bar{p}_j, t_j), \cdots, (\widetilde{p}_n, t_n) \rangle, \tag{C.2}$$

is composed of a sequence of map-matched points (e.g., $\widetilde{p}_i$) and missing points (e.g., $\hat{p}_j$).

In the rest of the paper, we will use the terms *map-matched point* (*map-matched ϵ-sampling-rate trajectory*) and *point* (*trajectory*) interchangeably when the context is clear.

Definition 31 (Platform Center) *A platform center C, e.g., the distributed data center of a company, keeps its map-matched trajectory dataset $\mathcal{T} = \{T_1, T_2, \cdots, T_l\}$, where T_i is the i-th map-matched trajectory and l is the length of $\mathcal{T}$. Note that we consider a platform as a client.*

Based on the above definitions, we formally define the studied problem as follows.

Federated Trajectory Recovery Problem Statement. Given a server $\mathcal{S}$ and N clients with their local trajectory datasets $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \cdots, \mathcal{T}_N\}$, each dataset $\mathcal{T}_i$ $(1 \leq i \leq N)$ in client C_i is a set of incomplete map-matched trajectories, i.e., $\mathcal{T}_i = \left\{T^i_{icp,1}, \cdots, T^i_{icp,l}\right\}$. Our problem aims to learn a shared global function $F(\cdot)$, such that for any incomplete map-matched trajectory, their missing coordinates are recovered. Formally, for each incomplete trajectory T_{icp} in a platform center, we have

$$\overbrace{\langle \cdots, (\bar{p}_i, t_i), \cdots, (\widetilde{p}_j, t_j), \cdots \rangle}^{\text{Incomplete Trajectory}} \xrightarrow{F(\cdot)} \overbrace{\langle (\cdots, (\hat{p}_i, t_i), \cdots, (\widetilde{p}_j, t_j), \cdots \rangle}^{\text{Recovered Trajectory}}, \tag{C.3}$$

where $1 \leq i < j \leq n$.

3 Analysis and Observations

In this section, we first analyze existing trajectory modeling proposals in a generic framework and investigate the complexity of representative models. Next, we identify directions for achieving a decentralized lightweight trajectory recovery framework.

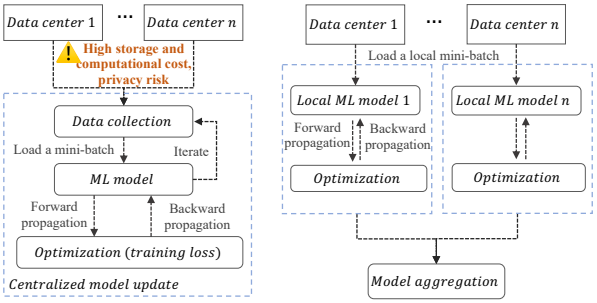


Fig. C.2: (a) A generic framework of existing trajectory recovery methods. (b) An illustration of the federated learning framework.

To gain insight into the prospects of the decentralized lightening trajectory modeling methods, we consider representative models [7, 10–12, 14, 27]. Figure C.2(a) shows a generic framework for these models. Generally, a trajectory recovery model has three components:

1. *a data collection module* that gathers data collected from different data centers into a central server;

2. *a machine learning (ML) model* consisting of stacked spatio-temporal blocks to extract spatio-temporal correlations as high-dimensional features;

3. *an optimization module* that iteratively updates the model with loaded mini-batch data by means of forward and backward propagation.

We observe that these trajectory recovery models are trained in a centralized manner while fail to consider the decentralized setting. Although models learned on such centralized trajectory data hold the promise of greatly improving usability, there are risks and responsibilities to storing trajectories in a centralized location due to their sensitive nature, which also leads to high storage and computational cost [16]. In addition, it is dangerous that real data can be used by a malicious third party. Thus, people may be unwilling to disclose their raw trajectories, which are especially sensitive and private.

This calls for a decentralized trajectory learning model where the data and computation are distributed among multiple data centers, which brings the benefit of privacy protection and potentially more efficient processing. We thus propose to design a new federated trajectory recovery framework, to be detailed in Section 4.2. Figure C.2(b) shows the typical learning process of a federated learning framework. Specifically, each data center trains a local model using its private data, while the server aggregates the parameters of the learned local models periodically to collaboratively obtain a global model.

In addition, we discuss the benefits in privacy protection of the proposed framework LightTR+ briefly. First, LightTR+ alleviates sensitive information leakage since it keeps the training data decentralized and only sends model parameters to the server. LightTR+ also mitigates malicious attacks by distributing the learning process across clients, making it harder for an untrusted third party to use the real trajectories.

In this study, we only consider deep learning (DL) based trajectory modeling solutions, which usually are composed of a stack of spatio-temporal (ST) blocks. ST-blocks, including stacked *ST-operators*, are the basic ingredients for extracting comprehensive features. We categorize popular *ST-operators* into different families based on the base operators that they extend: *convolutional neural network (CNN) based ST-operators, recurrent neural network (RNN) based ST-operators, and attention (Attn) based ST-operators*. A line of studies [13, 28] applies *CNN-based ST-operators*, i.e., temporal convolutional networks, that applies dilated causal convolutions to trajectory data for feature extraction. Another line of research [11, 14] uses *RNN-based ST-operators* that process trajectories based on a recursive mechanism, which is usually combined with graph neural network. The rest of studies [10, 12] adopts *Attn-based ST-operators* that enables weighted temporal information extraction over long sequences by applying the attention mechanism to establish self-interactions of input trajectories. The time and space complexity of the base operators is reported in Table C.1, where D is the number of embedding dimensions, N is the number of trajectories, and L is the maximum length of historical trajectories (each historical trajectory may have different length).

Table C.1: Categorization and Analysis of Base *ST-operators*.

Categorization		Time Complexity	Space Complexity
ST-operator	CNN [13, 28]	$O(D^2 \cdot N \cdot L)$	$O(D^2)$
	RNN [11, 14]	$O(D^2 \cdot N \cdot L)$	$O(D^2)$
	Attn [10, 12]	$O(D^2 \cdot N \cdot L \cdot (D + L))$	$O(D^2)$

As shown in Table C.1, we observe that ST-operators, which are the main components of trajectory learning models, incur high computational and storage costs, where the time and space complexities are both proportional to D^2. *It motivates us to develop lightweight* ST-operators *for trajectory recovery.* Although it is straightforward for achieving lightness to manipulate the embedding size D of *ST-operators*, many studies [29–31] have shown that reducing D will inevitably degrade learning capabilities of *ST-operators*.

Recent studies [32, 33] show the superior capabilities of multi-layer perceptron (MLP) (also called Dense) based architectures in feature extraction. MLP-based architectures benefit in low computational cost and fewer parameters while keeping comparable performance to mainstream *ST-operators*. Thus, we propose to simplify and reduce the model computations by replacing a majority of the base network (e.g., CNN) in *ST-operators* with MLP.

4 Methodology

In this section, we propose a federated trajectory recovery framework, entitled LightTR+, as shown in Figure C.3.

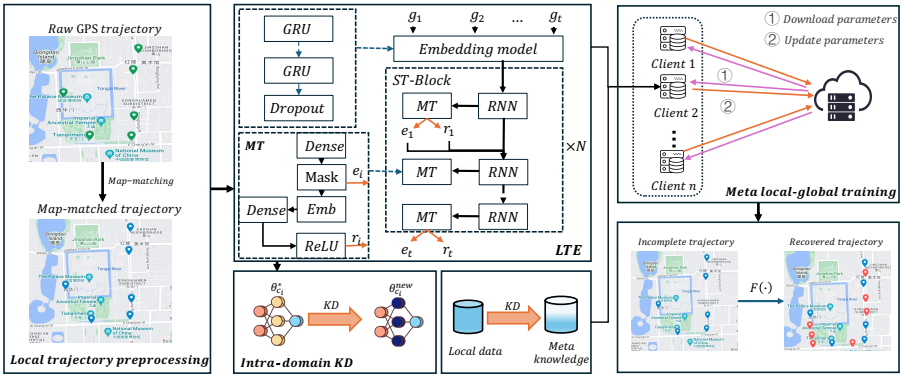


Fig. C.3: LightTR+: Lightweight Trajectory Recovery Framework

4.1 Framework Overview

As shown in Figure C.3, LightTR+ performs horizontal federated learning, i.e., a server-clients architecture, to collaboratively train local trajectory learning models under the orchestration of a central server, while keeping the local data privacy. LightTR+ can be divided into three modules: local trajectory preprocessing and light embedding, intra-domain knowledge distillation (KD), and meta-knowledge enhanced local-global training.

- *Local Trajectory Preprocessing and Light Embedding.* In this module, each client first preprocesses the collected trajectory data to generate map-matched trajectories using DHN [26]. Moreover, we introduce a lightweight local trajectory embedding module to reduce computation costs. Finally, each client downloads parameters and learns a local model to recover trajectories.
- *Meta-knowledge Enhanced Local-Global Training.* In this module, the central server aggregates the updated parameters from selected clients to obtain a global model. In particular, a novel meta-knowledge enhanced local-global training method is proposed to accelerate model convergence and reduce the cost of communication between clients and the server by means of knowledge distillation. In addition, the learned meta-knowledge contends with inter-domain interference across clients.
- *Intra-Domain Knowledge Distillation.* In this module, we first train a local model on private trajectory data for each client. When new trajectory data arrives, we employ a distillation loss to transfer knowledge of the previously learned local model to the updated local model learned on the new data.

Next, we will provide the technical details of each module, respectively.

4.2 Local Trajectory Preprocessing and Light Embedding

In this section, we report the preprocessing of local trajectories and the proposed lightweight trajectory embedding module in a client. Note that the model for the server has the same trajectory learning model architecture, to be detailed in Section 4.2, as each clients.

Local Trajectory Preprocessing

Given a road network, we first map all collected trajectories into the corresponding road intersections by converting their GPS location into discrete

units, referring to the Hidden Markov Model (HMM) in DHN [26]. Specifically, given a low-sampling-rate trajectory τ, we obtain the corresponding map-matched trajectory T as follows:

$$\begin{aligned} \tau &= \langle (p_1, t_1), \cdots, (p_i, t_i), \cdots (p_n, t_n) \rangle \\ T &= \langle (g_1, t_1), \cdots, (g_i, t_i), \cdots (g_n, t_n) \rangle = HMM(\tau), \end{aligned} \tag{C.4}$$

where $g_i = (x_i, y_i, tid_i)$, $\forall 1 \leq i \leq n$ is the converted unit, and x_i and y_i represents the i-th grid cell. We extract $tid_i = \lfloor \frac{t_i - t_0}{\epsilon} \rfloor$ to guide the model to learn how many points should be recovered between two consecutive low-sampling-rate points.

Lightweight Trajectory Embedding

As shown in Figure C.3, the Lightweight Trajectory Embedding (LTE) module is composed of an embedding model and stacked ST-blocks. The embedding model encodes the map-matched trajectory T into a single vector to capture its complex sequential dependencies, while the ST-blocks, containing a customized lightweight *ST-operator*, aim to predict the road segment e and moving ratio r of missing points, simultaneously.

We proceed to present the *embedding model* and the lightweight *ST-blocks*.

Embedding Model. Given a map-matched incomplete trajectory $T_{icp} = <(\widetilde{p_1}, t_1), \cdots, (\widetilde{p_i}, t_i), \cdots, (\widetilde{p_n}, t_n)>$, the embedding model aims to convert T into a single vector to capture the complex spatial and temporal correlations. It is important to model the sequential dependencies of trajectories, which enables accuracy [12]. We adopt the Gated Recurrent Unit (GRU) architecture for the embedding layer since GRUs bring the benefits of efficient long-term temporal dependency capture without performance decay. We input the map-matched raw trajectory T into the GRU to obtain its hidden features, which are then input into the ST-blocks. Specifically, we pass the input trajectory T along the time span $[1, 2, \cdots, n]$ through the GRU. For each time step t, the hidden features can be formulated as follows,

$$\begin{aligned} r_t &= \sigma(W_r \cdot [h_{t-1}, g_t] + b_r) \\ z_t &= \sigma(W_z \cdot [h_{t-1}, g_t] + b_z) \\ \tilde{h_t} &= tanh(W_h \cdot [r_i * h_{t-1}, g_t] + b_h) \\ h_t &= (1 - z_t) * h_{t-1} + z_i * \tilde{h_t}, \end{aligned} \tag{C.5}$$

where W represents the weight for respective gate neurons and b is the bias for the respective gate, and $[\cdot]$ denotes the feature concatenation.

For simplicity, the embedding model derives the hidden features h_t of the encoder for low-sampling-rate trajectories embedding:

$$h_t = embedding(h_{t-1}, s_{t-1}), \tag{C.6}$$

where the last state h_t is considered as the learned hidden features, which is also used as the input of ST-blocks.

ST-blocks. The learned hidden features h_t are then input into the stacked ST-blocks to obtain the data representations for incomplete trajectory recovery. As illustrated in Figure C.3, to increase scalability and reduce computational costs, we design a lightweight *ST-operator* including an RNN layer, where each RNN cell is followed by a pure MLP-based multi-task (MT) model, which benefits from the low space and time complexities of MLPs. The RNN can only predict the numerical coordinates of trajectories; it cannot ensure that the predicted missing points fall onto the road network. Thus, the lightweight *ST-operator* contains a novel MT model to predict the road segment e and moving ratio r at the same time due to their high correlations, leveraging multi-task learning [34].

More specifically, we feed h_t into the RNN to obtain the high-dimensional hidden features h'_t, which capture the sequential dependencies. Next, h'_t is input into the MT model to predict the road segment e_t and moving ratio r_t simultaneously. The lightweight *ST-operator* is formulated as follows.

$$\begin{aligned} h'_t &= RNN(h_t) \\ e_t, r_t &= MT(h'_t) \end{aligned} \tag{C.7}$$

The left center in Figure C.3 shows the detailed architecture of the MT model to predict e_t and r_t, simultaneously. Specifically, after obtaining hidden features through the RNN, we first use a dense (MLP) layer with a constraint mask layer (Mask) and a road segment embedding layer (Emb) to predict the road segment e_t, which is formulated as follows.

$$\begin{aligned} h_{t,d} &= Dense(h'_t, W_d) = W_d \cdot h'_t + b_d \\ e^t &= Mask(h_{t,d}) \\ h_{t,e} &= Emb(h_{t,d}, e^t) = ReLU(h_{t,d} + RNN(e^t)) \\ r_t &= ReLU(Dense([h_{t,e}, e^t], W_r)) = W_r \cdot [h_{t,e}, e^t] + b_r, \end{aligned} \tag{C.8}$$

where W and b denote the trainable parameters and bias, respectively.

Then, e_t together with h'_t are concatenated to go through the dense layer with an activation function (i.e., ReLU) to predict the moving ratio r_t. Next, e_t and r_t affect e_{t+1} and r_{t+1}. For example, people may be more willing to visit places that are close to their current locations. We also input e_t and r_t into the RNN. When inferring the missing points between two consecutive points of a low-sampling-rate trajectory, the hidden state h_t of our ST-blocks can be represented as follows.

$$h_t = STBlocks(h_{t-1}, e_{t-1}, r_{t-1}), \tag{C.9}$$

where e_{t-1} and r_{t-1} represent the road segment embedding of the predicted road segment and its moving ratio at timestamp $t-1$, respectively.

Constraint Mask Layer. In order to reduce the training complexities and achieve fine-grained trajectory recovery, we employ a constraint mask layer [11]. We use an exponential function (shown in Eq. C.10) to capture the influence of distance.

$$c_i = exp(\frac{-dist^2(p_i, \widetilde{p_i})}{\gamma}), \tag{C.10}$$

where $dist(p_i, \widetilde{p_i})$ denotes the Euclidean distance between the original point p_i and its map-matched point $\widetilde{p_i}$ on the corresponding road segment e and γ is a parameter related to the road network (γ = 125 in our work). Note that we only consider points in low-sampling-rate trajectories that are not far away from road segments. If the distance between a road segment e and point p_i is more than 50 meters, we set $\omega(e, p_i)$ to 0. Finally, we combine Eq. C.10 with softmax as the constraint mask layer, expressed as follows.

$$P(e_i|h_i) = \frac{exp(h_{t,d}^T \cdot w_c) \odot c_i}{\sum_{c' \in C} exp(h_{t,d}^T \cdot w_{c'}) \odot c_i}, \tag{C.11}$$

where w_c is a trainable parameter matrix. We then use $argmax$ to get the final prediction of road segment e^t.

Intra-Domain Knowledge Distillation

To alleviate intra-domain interference and catastrophic forgetting, we propose an intra-domain knowledge distillation module that enables effective knowledge transfer from a model learned on previous trajectory data to an updated model learned on incoming trajectory data. To enable streaming learning, we save the local trajectory model learned on previous local data and use it to constrain subsequent local updating with a distillation loss.

In particular, we perform two training steps for each client. First, we train a local model $\theta_{c_i}^*$ on client c_i's private trajectory data by means of the proposed local trajectory embedding model. Second, when new data arrives, we finetune the current local model $\theta_{c_i}^*$ with the new data to obtain a new local model $\theta_{c_i}^{new}$. We apply a distillation loss $L_{distill}$ between $\theta_{c_i}^{new}$ and $\theta_{c_i}^*$ to facilitate effective knowledge transfer from the previously trained data to the new data in order to avoid intra-domain interference. The distillation loss is designed based on knowledge distillation, where we consider $\theta_{c_i}^*$ and $\theta_{c_i}^{new}$ as the teacher and a student model, respectively. This is formulated as follows.

$$L_{distill} = softmax(\theta_{c_i}^*) log \frac{softmax(\theta_{c_i}^*)}{softmax(\theta_{c_i}^{new})} \tag{C.12}$$

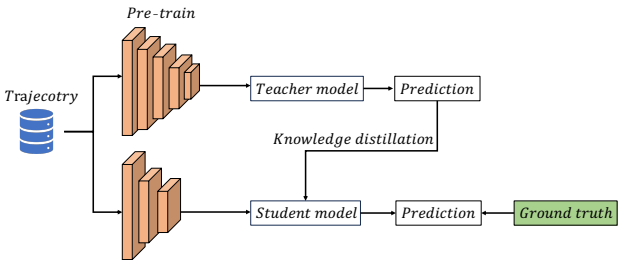


Fig. C.4: The Process of Knowledge Distillation

4.3 Meta-knowledge Enhanced Local-Global Training

To contend with inter-domain interference and reduce the communication cost, we propose a meta-knowledge enhanced local-global training (or meta local-global training for short) module for model aggregation by means of knowledge distillation. Figure C.4 shows the process of knowledge distillation, which includes a teacher model and a student model. Specifically, we first pre-train a teacher model, also called a meta-learner, to learn meta-knowledge for each client with a subset of local trajectories (see Algorithm 9). During local training, each client first downloads parameters from the server, and the learned meta-knowledge is used to guide the local model (i.e., lightweight trajectory embedding) learning. We consider the local model as the student model. Finally, the server aggregates parameters to update the global model with a subset of clients. In this study, we use the lightweight trajectory embedding model as the student and teacher model.

In federated trajectory recovery, we have N clients, denoted as $\{C_1, C_2, \cdots, C_N\}$, with their own datasets $\{\mathcal{T}_1, \mathcal{T}_2, \cdots, \mathcal{T}_N\}$. Each dataset $\mathcal{T}_i$ consists of three parts, a training, validation, and test dataset, denoted as $\mathcal{T}_i^{train}$, $\mathcal{T}_i^{valid}$, and $\mathcal{T}_i^{test}$, respectively. We aim to learn a good trajectory recovery model $f_i(\cdot)$ for each client on its local dataset $\mathcal{T}_i$:

$$min \frac{1}{N} \sum_{i=1}^{N} \frac{1}{n_i^{test}} \sum_{j=1}^{n_i^{test}} L_{total}(f_i(T_{icp}^j), T_{ground}^j), \tag{C.13}$$

where L_{total} is the overall loss function, n_i^{test} is the length of $\mathcal{T}_i^{test}$, and T_{ground} is the ground truth of complete trajectory.

We proceed to introduce the local trajectory recovery objective function and meta-knowledge enhanced local-global training.

Local Trajectory Recovery Objective

As we aim to predict road segment and moving ratio simultaneously, we combine the cross-entropy loss $L_1(\theta)$ and mean squared error $L_2(\theta)$ as the local

loss function $L_{local}(\theta)$ for trajectory recovery. $L_1(\theta)$ is used for road segment prediction, while we use $L_2(\theta)$ for moving ratio prediction.

$$L_{local}(\theta) = L_1(\theta) + \mu L_2(\theta), \tag{C.14}$$

where μ is a tunable parameter to balance the trade-off linearly between the road segment and moving ratio prediction. To enable streaming learning, we save the local trajectory model learned on previous local data, which is adopted to constrain the subsequent local updating with a distillation loss. Specifically, when new trajectory data arrives, we employ an additional distillation loss $L_{distill}$ to obtain a local loss function $L_{local}(\theta)$, formulated as follows.

$$L_{local}(\theta) = L_1(\theta) + \mu L_2(\theta) + L_{distill}, \tag{C.15}$$

where $L_{distill}$ is introduced to alleviate the catastrophic forgetting incurred by intra-domain interference.

The cross-entropy loss is formulated as follows.

$$\begin{aligned} L_1(\theta) = - \sum_{(T,\tilde{T}\epsilon\mathcal{T})} \sum_{j=1}^{|\tilde{T}|} \sum_{l=1}^{L} 1a_j \cdot e_l log(R_\theta(\hat{a}_j e = e_l | d_1 : j-1)) \\ s.t. d_{j-1} = (T, \tilde{T}_{1:j-1}), \end{aligned} \tag{C.16}$$

where T and $\tilde{T}$ represent incompleted trajectories and recovered completed trajectories, respectively, L is the number of road segments, $a_j e$ is the ground truth of road segment, $\hat{a}_j$ is the prediction, and R_θ represents the lightweight trajectory embedding module. Next, D is the training data.

The mean squared error is also employed to predict the movement ratio, which can be formulated as:

$$\begin{aligned} L_2(\theta) = - \sum_{(T,\tilde{T}\epsilon\mathcal{T})} \sum_{j=1}^{|\tilde{T}|} (a_j r - R_\theta(d_{j-1})^2) \\ s.t. \, d_{j-1} = (T, \tilde{T}_{1:j-1}), \end{aligned} \tag{C.17}$$

where $a_j r$ is the real moving ratio.

Meta-knowledge Enhanced Local-global Training

To achieve faster convergence and reduce communication costs, we propose a meta-knowledge enhanced local-global training module that uses knowledge distillation. We first randomly select a subset from the local dataset for each client and pre-train a meta-learner (a teacher model) to accumulate meta-knowledge across clients. By combining this with knowledge distillation, the learned meta-knowledge can alleviate data heterogeneities, thus mitigating

Algorithm 9 Teacher model training

Input: subsets of Dataset $\{\mathcal{T}_i\}_{i=1}^{N}$ at N clients, λ_0, l_t
Output: A common teacher model f_{tea}
1: $\lambda \leftarrow \lambda_0$;
2: Train local teacher model f_i using L_{local} with $\mathcal{T}_i^{train}$ at each client C_i;
3: Send the current model f_i to the next client C_{i+1};
4: $acc_{i+1}^{vaild} \leftarrow f_{i+1}(\mathcal{T}_{i+1}^{valid})$;
5: **if** $acc_{i+1}^{valid} > l_t$ **then**
6: Train f_{i+1} with $\mathcal{T}_{i+1}$ using Eq.C.19;
7: **else**
8: $f_i \leftarrow f_{i+1}$
9: Train f_{i+1} with $\mathcal{T}_{i+1}$ using Eq.C.19;
10: **end if**
11: Repeat steps 3 – 10 until convergence;
12: $f_{tea} \leftarrow f_N$;
13: **Return** f_{tea}

inter-domain interference across clients. Each client trains a local model after downloading parameters from the server with meta-knowledge for guiding the local training. The server obtains a global model by aggregating parameters from the participating clients.

Algorithm 9 specifies the process of teacher model training, which is used by each clients. Here, we train all clients sequentially in a cyclic way, transfering previous knowledge to the next. The previously learned knowledge is preserved if it is useful for the current client training. Otherwise, it is discarded. This alleviates data heterogeneity across clients. We fix $\lambda = \lambda_0$ to preserve suficient common knowledge (line 1). Threshold l_t is used to determine whether to completely preserve the previously learned knowledge (lines 2–10). When accuracy $acc_{i+1}^{valid} < l_t$, too little information is contained in the current client C_{i+1}. We directly initialize the current model with the previous one to fully leverage previous knowledge, and thus handle the data heterogeneity.

We employ the proposed lightweight trajectory embedding model as the teacher model f_{tea} to perform the trajectory missing points prediction task with incomplete trajectories. The teacher model is then used to guide the training of the local model f_{stu} via knowledge distillation. The knowledge distillation is formulated as follows.

$$L_{dist}(f_{tea}, f_{stu}; T_{icp}) = ||f_{tea}(T_{icp}) - f_{stu}(T_{icp})||_2^2, \tag{C.18}$$

where $||\cdot||_2$ is the $L2\text{-}norm$.

Through knowledge distillation, we can make good use of meta-knowledge to guide the local training for each client, which alleviates data heterogeneities

across clients. Therefore, the total loss L_{total} function for each client C_i is:

$$L_{total} = \frac{1}{|\mathcal{T}^{train}|} \sum_{(T_{icp}, T_{ground}) \epsilon \mathcal{T}_i^{train}} L_{local}(R_\theta(T_{icp}), T_{ground}) + \lambda L_{dist}(f_{tea}, R_\theta; T_{icp}), \tag{C.19}$$

where λ is a trade-off knowledge transfer and focusing on the current data, and L_{local} is the trajectory recovery loss.

Algorithm 10 Meta-Knowledge Enhanced Local Training

Input: Datasets $\{\mathcal{T}_i\}_{i=1}^N$ in N clients, λ_0, l_t, teacher model f_{tea}, training epochs $epoch_{max}$

Output: Local Trajectory Recovery models $\{R_\theta^i\}_{i=1}^N$

1: $\lambda \leftarrow 0$;
2: $c_n \leftarrow 0$;
3: **while** $c_n < N$ **do**
4: $\quad acc_i^{train} \leftarrow R_\theta^i(\mathcal{T}_i^{train})$;
5: $\quad$ Train R_θ^i using Eq. C.19;
6: $\quad acc_{i,tea}^{valid} \leftarrow f_{tea}(\mathcal{T}_i^{valid})$;
7: $\quad acc_i^{valid} \leftarrow R_\theta^i(\mathcal{T}_i^{valid})$;
8: $\quad$ **if** $acc_{i,tea}^{valid} \leq acc_i^{valid}$ and $acc_i^{valid} < l_t$ **then**
9: $\quad\quad \lambda \leftarrow 0$;
10: $\quad$ **else**
11: $\quad\quad \lambda \leftarrow \lambda_0 \cdot 10^{min(1,(acc_{i,tea}^{valid} - acc_i^{valid})*5)-1}$;
12: $\quad$ **end if**
13: **end while**
14: Repeat steps 3 – 15 in $epoch_{max}$ iterations;
15: **Return** $\{R_\theta^i\}_{i=1}^N$

Algorithm 10 illustrates the process of meta-knowledge enhanced local training (i.e., knowledge distillation) to train a good local trajectory recovery model for upcoming central aggregation. Specifically, we first set λ to 0, which means no knowledge is learned from the teacher model (line 1). For each client, we apply gradient descent to update its local model R_{θ_i} with local training data using Eq. C.19 (lines 3–5). When the accuracy $acc_{i,tea}^{valid}$ of the teacher model is below acc_i^{valid} of the current local model on local validation data, we keep λ at 0, meaning that there is no guidance by the teacher model as it has less knowledge of the current client, and we do not want to rely on it. Otherwise, we dynamically update λ as follows.

$$\lambda \leftarrow \lambda_0 \cdot 10^{min(1,(acc_{i,tea}^{valid} - acc_i^{valid})*5)-1}, \tag{C.20}$$

Eq. C.20 shows that the better the teacher model performs, the larger the value of λ is, which means that the local model learns more knowledge from the

Algorithm 11 Local-global Parameter Updating

Input: communication rounds R, initial global model of server θ_s^0, number of clients N, learning rate α, local training epochs E
Output: final global model of server θ_s

1: **for** each round r in $[1, 2, \cdots, R]$ **do**
2: Randomly select $C \in [1, N]$ clients from N clients;
3: **for** each client $c_i \in C$ **do**
4: The server transmits θ_s^{r-1} to client c_i;
5: **for** each local epoch e in $[1, 2, \cdots, E]$ **do**
6: $\theta_{c_i}^r \leftarrow argmin_\theta L_{total}(\theta_{c_i}^r)$;
7: $\nabla L_{total}(\theta_{c_i}^r) \leftarrow \nabla L_{total}(\theta_{c_i}^{r-1}) - \alpha(\theta_{c_i}^r - \theta_{c_i}^{r-1})$;
8: **end for**
9: $\theta_{c_i} \leftarrow \theta_{c_i}^r$;
10: **end for**
11: $\theta_s \leftarrow \sum_{c_i \in C} \frac{1}{C} \theta_{c_i}$;
12: **end for**

return θ_s

Table C.2: Performance of Training on Streaming Data on Two Datasets (Keep Ratio = 12.5%)

Dataset	Method	M1 → Geolife		M1 → Tdrive		M2 → Geolife		M2 → Tdrive	
		Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision
Geolife	**MTrajRec+FL**	0.587	0.611	0.087	0.096	0.323	0.344	0.187	0.231
	LightTR	0.677	0.743	0.095	0.112	0.455	0.472	0.228	0.285
	LightTR+	0.695	0.724	0.108	0.134	0.511	0.546	0.311	0.437
TDrive	**MTrajRec+FL**	0.092	0.112	0.510	0.544	0.336	0.375	0.208	0.221
	LightTR	0.112	0.154	0.611	0.668	0.377	0.436	0.255	0.337
	LightTR+	0.146	0.114	0.608	0.661	0.522	0.612	0.410	0.487

teacher model. After several iterations of model updating, we obtain N local models for the preceding central aggregation (lines 5–15).

Algorithm 11 shows the process of local-global parameter updating. At each communication round r, we randomly select a subset of N clients C, which further reduces communication costs. The server transfer its current model θ_s^{r-1} to selected clients (lines 1–4). Each client performs meta-knowledge enhanced local training according to Algorithm 10, which optimizes a local empirical risk objective, i.e., Eq. C.19 (lines 5–10). Next, the selected clients upload their local model parameters θ_{c_i} to the server. Finally, the server aggregates the uploaded parameters to update the global model θ_s until it converges to a stationary point θ_s (line 11).

5 Experimental Study

5.1 Setup

Datasets

The experiments are carried out on two real-world public trajectory datasets: Tdrive and Geolife.

- **Tdrive.** The Tdrive dataset contains taxi-trip-based trajectories in Beijing from February 2, 2008, to August 2, 2008, including roughly 15 million trajectory points.
- **Geolife.** The Geolife dataset contains 17621 taxi GPS trajectories in Asia from April 2007 to August 2012, including 182 taxi drivers.

Evaluation Metrics

We aim to recover low-sampling-rate trajectories in free space to high-sampling-rate trajectories mapped onto the road network. Thus, both accuracy of road segment recovery and distance error of location inference are adopted to compare the performance of our model and baselines.

Recall & Precision. We use recall and precision to evaluate the performance of route recovery by comparing the recovered road segments P_R to the corresponding ground truths G. Recall and precision are defined as follows.

$$Recall = \frac{|P_R \cap G|}{|G|}, Precision = \frac{|P_R \cap G|}{|P_R|} \tag{C.21}$$

MAE & RMSE. The distance measures, i.e., Mean Absolute Error (MAE) and Root Mean Square Error (RMSE), are used to evaluate the point recovery performance. Note that we calculate the distance error based on the road network by updating the Earth distance to road network constrained distance. The smaller the MAE and RMSE are, the better the models perform. MAE and RMSE are defined as follows.

$$\begin{aligned} MAE &= \frac{1}{m}\sum_{j=1}^{m}|dis(g_j, \hat{g}_j)|, \\ RMSE &= \sqrt{\frac{1}{m}\sum_{j=1}^{m}(|dis(g_j, \hat{g}_j)|)^2} \\ s.t. dis(g_j, \hat{g}_j) &= min(rn_{dis}(g_j, \hat{g}_j), rn_{dis}(\hat{g}_j, g_j)), \end{aligned} \tag{C.22}$$

where g_j is the ground truth location, $\hat{g}_j$ is the predicted map-matched trajectory point, and $rn_{dis}(g_j, \hat{g}_j)$ is the length of the shortest path between the

prediction and the ground truth. Next, we use $min(rn_{dis}(g_j, \hat{g}_j), rn_{dis}(\hat{g}_j, g_j))$ as the final error since the road network is a directed graph, where the distance from g_i to g_j may not be equal to the distance from g_j to g_i.

Baseline

We compared the proposed LightTR+ with the following baselines. To the best of our knowledge, there is currently no FL-based method for discrete trajectory recovery. To enable fair comparisons, we transfer existing centralized baselines for trajectory recovery to their federated versions by combining them with FedAvg [16].

- **FC+FL.** The FC+FL method combines horizontal FL with stacked fully connected layers (FC) [35] for trajectory recovery.
- **RNN+FL.** The RNN+FL method is a decentralized trajectory recovery model, where stacked RNNs are integrated with horizontal FL to collaboratively learn trajectory representations.
- **MTrajRec+FL.** The MTrajRec+FC devises a decentralized trajectory recovery model, where the MTrajRec [11] is employed as the local model.
- **RNTrajRec+FL.** The RNTrajRec+FL method is a horizontal FL trajectory recovery model, where we set the local model to be RNTrajRec [22], which applies a graph neural network to learn rich trajectory representations.
- **LightTR.** The LightTR method is a state-of-the-art federated trajectory recovery model with a lightweight network, which we proposed in our preliminary study [21].

Implementation Details

We implement the new model with the Pytorch framework on a GPU server with an NVIDIA GTX 1080Ti GPU. The parameters in the model are set as follows. We set training epochs to 50 for each client. The initial learning rate is 0.001. The hidden features dimensionality in the local model is set to 512. To prevent overfitting, we set the dropout ratio to 0.5 in the embedding module. In addition, the number of clients is set to 20 by default, and each client owns its decentralized data.

Data Preprocessing

We split the datasets into training, validation, and testing sets with a splitting ratio of 7:2:1. The datasets contain high-sampling-rate trajectories. We thus randomly remove points to obtain low-sampling-rate trajectories with a

keep ratio. In this study, we use the keep ratio 6.25%, 12.5%, and 25%. Six points between each two consecutive points in an incomplete trajectory are required to be restored on average. Referring to an existing study [11], we apply HMM on the original trajectories to get the ground truth for subsequent model performance comparisons.

5.2 Experimental Results

Performance on Training on Streaming Data

To evaluate the performance of training on streaming data, two representative baselines, MTrajRec+FL and LightTR, are compared with LightTR+. We first train an initial model M1 on either Geolife or Tdrive and then fine-tune M1 with the other dataset to obtain an updated model M2. The results are shown in Table C.2, where M1 (or M2) → Geolife (or Tdrive) means that applying M1 (or M2) on Geolife (or Tdrive) for testing. The overall best performance is marked in bold. As the performance on both datasets exhibits a similar trend, for brevity, we only show the observations when M1 is trained on Geolife.

We first apply M1 on Geolife (M1 → Geolife) and Tdrive (M1 → Tdrive), simultaneously. LightTR+ consistently performs better than the two baselines. MTrajRec+FL and LightTR offer acceptable Recall and Precision on Geolife, but their performance deteriorates on Tdrive. LightTR+ performs better than the two baselines on Tdrive in most cases. This may be because the proposed intro-domain knowledge distillation module enables more generalized feature learning.

M2 is obtained by fine-tuning M1 with Tdrive. After fine-tuning, we apply the updated model M2 on Geolife (M2 → Geolife) and Tdrive (M2 → Tdrive) simultaneously for testing. We observe that LightTR+ achieves the best results among the methods, which is evidence that LightTR+ can perform effective feature learning on new data while preserving old knowledge. This is because of the intra-domain knowledge distillation module, which avoids catastrophic forgetting by enabling effective knowledge transfer from previously learned data to current data. Simple fine-tuning based methods are unable to handle streaming data, as they fail to overcome both inter- and intra-domain interference. LightTR+ achieves relatively stable performance on both Geolife and Tdrive, demonstrating its strength over baselines. In addition, both LightTR and LightTR+ consistently outperform MTrajRec+FL due to the powerful representation learning capability of the proposed *ST-operator*.

Baseline Comparison

Table C.3 shows the performance comparison among different methods on the two datasets. The observations are as follows.

Table C.3: Overall Performance Comparison on Two Datasets

Datasets	Baseline	6.25%				12.5%				25%			
		Recall	Precision	MAE	RMSE	Recall	Precision	MAE	RMSE	Recall	Precision	MAE	RMSE
Geolife	**FC+FL**	0.212	0.219	0.755	1.015	0.283	0.215	0.728	1.007	0.315	0.294	0.685	0.807
	RNN+FL	0.392	0.417	0.630	0.776	0.397	0.394	0.628	0.741	0.456	0.438	0.561	0.659
	MTrajRec+FL	0.595	0.601	0.462	0.634	0.599	0.623	0.455	0.636	0.634	0.673	0.410	0.546
	RNTrajRec+FL	0.594	0.618	0.455	0.617	0.631	0.655	0.431	0.611	0.669	0.702	0.398	0.501
	LightTR	0.674	0.679	0.375	0.464	0.724	0.748	0.335	0.432	0.758	0.739	0.323	0.435
	LightTR+	0.683	0.711	0.366	0.471	0.701	0.722	0.354	0.449	0.705	0.728	0.336	0.424
Tdrive	**FC+FL**	0.211	0.198	0.753	1.001	0.283	0.232	0.723	1.022	0.276	0.270	0.662	0.804
	RNN+FL	0.358	0.355	0.584	0.750	0.392	0.396	0.547	0.682	0.456	0.464	0.536	0.664
	MTrajRec+FL	0.512	0.554	0.532	0.697	0.543	0.587	0.486	0.628	0.633	0.685	0.452	0.530
	RNTrajRec+FL	0.568	0.611	0.488	0.651	0.598	0.643	0.447	0.585	0.620	0.705	0.384	0.473
	LightTR	0.624	0.674	0.395	0.473	0.667	0.707	0.366	0.452	0.708	0.732	0.332	0.446
	LightTR+	0.611	0.677	0.402	0.497	0.671	0.698	0.365	0.448	0.687	0.701	0.355	0.473

- LightTR+ and LightTR achieve comparable results in most cases on the two datasets with different keep ratios (i.e., 6.25%, 12.5%, and 25%), and they outperform all the baselines. In addition, LightTR+ performs better than LightTR by up to 4.7% in terms of precision on Geolife when the keep ratio is 6.25%. Although LightTR outperforms LightTR+ on Tdrive with a 25% keep ratio, LightTR+ can avoid catastrophic forgetting when it is applied to streaming trajectories, showing its superiority in incremental trajectory recovery scenarios.

- LightTR performs better than the best among the baselines by up to 14.7% and 13.19% with keep ratio 12.5% in terms of Recall and Precision, respectively, on Geolife. This is because of the powerful feature learning capability of the proposed light *ST-operator*. Moreover, the learned meta-knowledge alleviates the data heterogeneity across clients, which further improves the model performance. We also observe that the performance improvements obtained by LightTR on Geolife exceed those on Tdrive since the Geolife data are more sufficient than Tdrive with more useful knowledge.

- All RNN-based methods (i.e., RNN+FL, MTrajRec+FL, and RNTrajRec+FL), which are able to learn temporal dependencies of trajectories, perform better than FC+FL on both Tdrive and Geolife. It shows the importance of effective temporal dependency capturing for trajectory embedding. FC+FL performs the worst on both datasets, which demonstrates that simply stacked FC layers are too shallow to capture the complex temporal correlations and thus shows the necessity of the newly proposed lightweight *ST-operator*.

- RNN+FL, MTrajRec+FL, and RNTrajRec+FL can capture the temporal dependencies, RNTrajRec+FL performs the best. This is because RNTrajRec is capable of learning more effective trajectory embeddings benefiting from the graph model, which incorporates the embedding of each road segment.

The above observations indicate that LightTR is more effective than the existing

Table C.4: Effect of Local Training Epoch on Both Datasets (keep ratio=12.5%)

Datasets	Metrics	Local Training Epoch			
		3	5	8	10
Geolife	**Recall**	0.533	0.622	0.686	0.701
	Precision	0.597	0.665	0.717	0.722
	MAE	0.499	0.432	0.366	0.354
	RMSE	0.554	0.511	0.452	0.449
Tdrive	**Recall**	0.547	0.611	0.667	0.671
	Precision	0.588	0.638	0.701	0.698
	MAE	0.502	0.443	0.377	0.365
	RMSE	0.611	0.533	0.475	0.448

Table C.5: Effect of the Number of Clients on Both Datasets (keep ratio=12.5%)

Datasets	Metrics	Number of Clients			
		5	10	15	20
Geolife	**Recall**	0.631	0.655	0.688	0.701
	Precision	0.643	0.711	0.742	0.722
	MAE	0.477	0.442	0.389	0.354
	RMSE	0.556	0.524	0.465	0.449
Tdrive	**Recall**	0.593	0.622	0.644	0.671
	Precision	0.628	0.673	0.688	0.698
	MAE	0.498	0.455	0.385	0.365
	RMSE	0.583	0.554	0.441	0.448

FL-based methods.

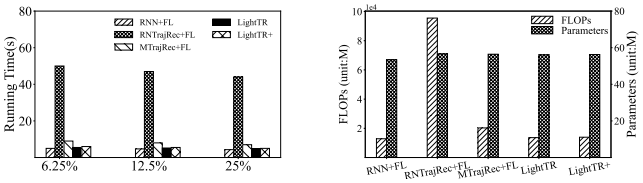
Effect of Local Training Epochs

We perform several rounds of local training in each client before global parameter aggregation. To test the effect of the number of local training epochs E on each client, we conduct experiments with 3, 5, 8, and 10 local training epochs. The results are shown in Table C.4. Generally, the results demonstrate that the model performance improves as the number of local training epochs increases. In addition, we see that training the local model for each client with more epochs is more likely to lead to better performance because more effective hidden features are learned with more local training epochs. Aggregating higher-quality local models leads to a superior global model, which enhances overall performance.

Effect of the Number of Clients

To study the effect of the number of clients involved in model training of the proposed LightTR+, we conduct experiments with 5, 10, 15, and 20 clients. The results are shown in Table C.5. We observe that the model performance gets better as the increase of number of clients, due to more clients bringing more training data. For example, the Recall on Geolife increases from 0.631 to 0.701, and the Precision on Tdrive increases from 0.628 to 0.698. One can also see that

the 20-client-based LightTR performs worse than the 15-client-based LightTR in terms of Precision on Geolife. Generally, the results demonstrate that more clients participating in training are more likely to lead to better performance because more useful knowledge is learned from more training data during the training.



(a) Running time (b) FLOPs and Parameters

Fig. C.5: Running Efficiency on Geolife

Table C.6: Effect of The Amount of Local Training Data (keep ratio=12.5%)

Datasets	Metrics	Fractions of Local Data			
		25%	50%	75%	100%
Geolife	**Recall**	0.204	0.435	0.603	0.701
	Precision	0.276	0.476	0.654	0.722
Tdrive	**Recall**	0.114	0.377	0.522	0.671
	Precision	0.156	0.413	0.576	0.698

Effect of the Amount of Local Traning Data

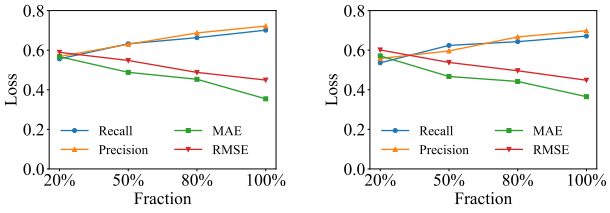
To study the effect of the amount of local training data involved in LightTR+ training, we conduct experiments with each client using 25%, 50%, 75%, and 100% of their local training data. The results are shown in Table C.6. We observe that increasing the amount of local training data yields improved model performance. It shows that incorporating more data into training is likely to introduce more valuable knowledge, resulting in a better-performing model.

Running Efficiency

Considering lightness and scalability, we study the running time (of each epoch), floating-point operations (FLOPs), and the number of parameters for LightTR+ with four RNN-based baselines (i.e., RNN+FL, MTrajRec+FL, RNTrajRec+FL, and LightTR) on Geolife. Figure C.5(a) shows the results of running time, while FLOPs and parameters consumptions are shown in Figure C.5(b). Generally, LightTR+ has significantly less running time and fewer

FLOPs due to the lightness of the proposed lightweight *ST-operator*. We also observe an exception in that RNN+FL achieves slightly less running time. However, RNN+FL is much less accurate. In addition, LightTR+ achieves faster overall convergence, which offers evidence that the proposed meta-knowledge enhanced local-global training module can accelerate model training. Specifically, LightTR+ converges after around 100 epochs, while MtrajRec+FL converges after around 160 epochs, on both datasets.

It is clear that LightTR+ uses fewer resources than the two most accurate competing models MTrajeRec+FL and RNTrajRec, while maintaining comparable accuracy. More specifically, LightTR+ reduces the running time by 90% and 32.9% compared with RNTrajRec+FL and MTrajRec, respectively. Additionally, LightTR+ obtains FLOPs reduction by 86.7% compared to RNTrajRec as the high calculation complexities of the attention mechanism in RNTrajRec. Note that the communication cost in FL is positively correlated with the number of model parameters and FLOPs [18, 36], where FLOPs are termed as system overhead. Thus, the fewer FLOPs and parameters consumed by LightTR+ bring the benefits of fewer communication costs and less system overhead in federated trajectory recovery. Overall, the results in Figure C.5 indicate the feasibility and scalability of LightTR+ for model deployment in real decentralized trajectory recovery scenarios.

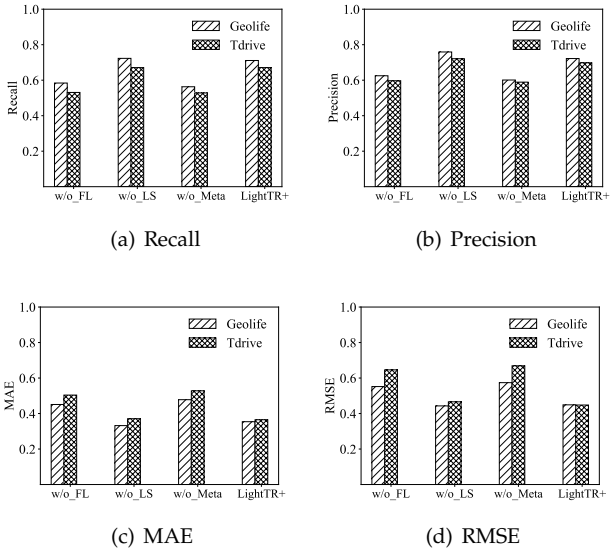


(a) Geolife (b) Tdrive

Fig. C.6: Effect of Client Fractions on Both Datasets (keep ratio=12.5%)

Effect of Client Fractions

When we aggregate parameters in LightTR, a certain fraction of clients is sampled to perform model training in each client. To study the effect of client fraction F on the model performance, we conduct experiments by sampling four different fractions of clients: 20%, 50%, 80%, and 100%, as shown in Figure C.6. Generally, the model performance exhibits an increasing trend with the increase in client fractions in terms of all evaluation metrics. LightTR+ achieved a most improvement of 26.1% in recall and 27.2% in precision, respectively, while getting MAE and RMSE most reduction by up to 37.6% and 27.8%. It indicates that the bigger fraction of clients with more training data is, the better model performs since more useful knowledge can be learned.



(a) Recall (b) Precision

(c) MAE (d) RMSE

Fig. C.7: Performance of LightTR and Its Variants on Both Datasets (keep ratio = 12.5%).

Ablation Study

To gain insight into the effects of key aspects of LightTR+, We evaluate three LightTR+ variants.

- **w/o horizontal FL (w/o_FL).** w/o_FL removes the central server of LightTR+, where each client trains their model locally and all clients exchange their local model to each other.
- **w/o Lightweight *ST-operator* (w/o_LS).** w/o_LS replace the Lightweight *ST-operator* with MTrajRec to learn local trajectory embeddings for each client.
- **w/o meta-knowledge enhanced local-global training module (w/o_Meta).** w/o_Meta replace the meta-knowledge enhanced local-global training module with FedAvg [16].

To assess whether the components in LightTR+ all contribute to the performance of trajectory recovery, we compare LightTR+ with its variants. The results are shown in Figure C.7. The observations are as follows.

- Regardless of the datasets, LightTR+ always performs better than its counterparts without horizontal FL, lightweight *ST-operator*, and meta-knowledge enhanced local-global training module. It shows that these components are all useful for trajectory recovery.
- LightTR+ performs slightly better than w/o_LS. This shows the powerful feature learning capability of the proposed lightweight *ST-operator*,

even compared with MTrajRec, a recent state-of-the-art trajectory recovery model. However, the complex model always has high space and time complexities, which incurs poor scalabilities and more computation resource requirements.

- w/o_Meta performs the worst among all variants, which shows the importance of the meta-knowledge enhanced local-global training module. It also offers evidence that the meta-knowledge enhanced local-global training module is capable of alleviating the data heterogeneities (intra-domain interference) across clients.

Parameter Sensitivities

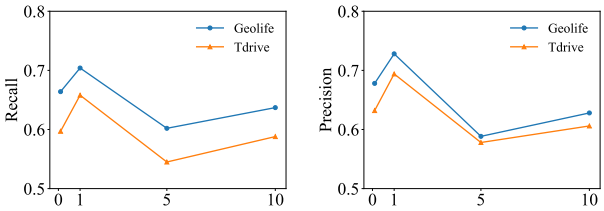
We next study how sensitive the model is on the hyper-parameter λ and the threshold l_t, as shown in Figure C.8. λ is set to different values, i.e., 0.1, 1, 5, 10. Figure C.8(a) illustrates the effect of λ, showing setting λ to 5 is the reasonable setting in this experiment. It indicates that the learned meta-knowledge plays a more important role in trajectory modeling, which shows the necessity of the proposed meta-knowledge enhanced local-global training module. Figure C.8(b) shows the effect of l_t, which is the threshold for knowledge accumulation, and we set it from 0 to 0.6. Here, we observe all curves slightly increase and then decrease. LightTR achieves the best result when setting l_t to 0.4. It indicates that excessive guidance from the meta-learner will make the local trajectory embedding model confusing, and thus degrade the trajectory recovery performance.

6 Related Work

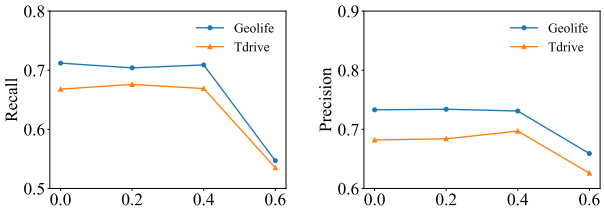
We briefly review prior studies on trajectory recovery and federated learning.

6.1 Trajectory Recovery

Trajectory recovery [7, 10, 11, 37] attracts increasing interest due to the increasing availability of trajectory data and rich downstream applications in intelligent transportation systems. Traditionally, trajectory recovery methods are mostly based on statistical models [7, 9]. However, the statistical models cannot capture complex temporal (and spatial) correlations of trajectory data due to their limited learning capacity. With the advance of deep learning techniques, recent deep learning based methods [10–12, 14, 21] address trajectory recovery by leveraging their powerful learning capabilities. For example, MTrajRec [11] addresses constrained trajectory restoration through Seq2Seq based multi-task learning. However, these methods are trained with



(a) Effect of λ



(b) Effect of l_t

Fig. C.8: Parameters Sensitivity Analysis on both datasets (keep ratio = 12.5%)

centralized data, while disregard decentralized settings and do not take computational efficiency into account.

6.2 Federated Learning

Federated Learning (FL) is a machine learning approach where many clients (often called edge devices) collaboratively train a model using decentralized data under the orchestration of a central server [15, 16]. Generally, FL can be divided into three categories: horizontal FL [16], vertical FL [38], and federated transfer learning [39]. Moreover, personalized FL [40, 41] is widely studied recently to address data heterogeneity across clients, which may degrade the performance of FL. FedDC [40] introduces local drift decoupling and correction to address Non-IID data across clients. However, most of the above methods are designed for computer vision, and cannot be applied to trajectory data due to the unique spatio-temporal patterns. Recently, FL is applied in spatio-temporal data (e.g., traffic data) to ensure data privacy [4, 42–44]. CNFGNN [4] is proposed to perform federated traffic prediction, which extracts spatial and temporal features utilizing GNN and GRU. However, these methods are computational heavily. There still lacks a well-customized FL model for trajectory recovery, considering the effectiveness and efficiency at the same time.

6.3 Incremental Learning

Incremental learning (IL), also called continuous learning, learns a sequence of tasks incrementally with knowledge transfer while mitigating catastrophic forgetting [18, 19]. IL aims to extend accumulated knowledge from an infinite stream of data for future learning. Traditionally, IL studies have focused on object recognition [45]. With the rapid development of deep learning techniques, deep learning based continuous learning methods [20, 46] have emerged, which can be divided into regularization-based [20], replay-based [46], and architcture-based [47] methods. A recent line of study [48, 49] combines federated learning with IL to ensure data privacy, termed FIL (Federated Incremental Learning). FIL can be categorized into synchronous FIL [49, 50] and asynchronous FIL [51, 52]. However, existing FIL and IL methods are primarily tailored for computer vision and natural language processing applications and do not extend to trajectory data due to its unique spatio-temporal characteristics.

7 Conclusion

We introduce LightTR+, a novel federated framework for lightweight incremental trajectory recovery that aims to support decentralized trajectory learning and privacy protection. To optimize computational efficiency, we propose a lightweight local trajectory embedding module with a lightweight *ST-operator* for each client, which effectively extracts features. We propose an intra-domain knowledge distillation module to alleviate intra-domain interference. In addition, a novel meta-knowledge enhanced local-global training module is proposed to offer improved communication efficiency between the server and the clients, thereby reducing data heterogeneity across clients and mitigating inter-domain interference. Extensive experiments on real data offer evidence that LightTR+ is capable of state-of-the-art accuracy at trajectory recovery while consuming fewer computational resources.

References

[1] S. Liu, H. Su, Y. Zhao, K. Zeng, and K. Zheng, "Lane change scheduling for autonomous vehicle: A prediction-and-search framework," in *SIGKDD*, 2021, pp. 3343–3353.

[2] M. Chen, Y. Zhao, Y. Liu, X. Yu, and K. Zheng, "Modeling spatial trajectories with attribute representation learning," *TKDE*, vol. 34, no. 4, pp. 1902–1914, 2022.

[3] L. Deng, H. Sun, Y. Zhao, S. Liu, and K. Zheng, "S2tul: A semi-supervised framework for trajectory-user linking," in *WSDM*, 2023, pp. 375–383.

[4] C. Meng, S. Rambhatla, and Y. Liu, "Cross-node federated graph neural network for spatio-temporal data modeling," in *SIGKDD*, 2021, pp. 1202–1211.

[5] Y. Tong, D. Shi, Y. Xu, W. Lv, Z. Qin, and X. Tang, "Combinatorial optimization meets reinforcement learning: Effective taxi order dispatching at large-scale," *TKDE*, pp. 1–1, 2021.

[6] K. Zheng, Y. Zheng, X. Xie, and X. Zhou, "Reducing uncertainty of low-sampling-rate trajectories," in *ICDE*, 2012, pp. 1144–1155.

[7] P. Banerjee, S. Ranu, and S. Raghavan, "Inferring uncertain trajectories from partial observations," in *ICDM*, 2014, pp. 30–39.

[8] J. Yuan, Y. Zheng, C. Zhang, X. Xie, and G.-Z. Sun, "An interactive-voting based map matching algorithm," in *MDM*, 2010, pp. 43–52.

[9] J. A. Alvarez-Garcia, J. A. Ortega, L. Gonzalez-Abril, and F. Velasco, "Trip destination prediction based on past GPS log using a hidden markov model," *Expert Systems with Applications*, vol. 37, no. 12, pp. 8166–8171, 2010.

[10] H. Sun, C. Yang, L. Deng, F. Zhou, F. Huang, and K. Zheng, "Periodic-move: Shift-aware human mobility recovery with graph neural network," in *CIKM'21*, 2021, pp. 1734–1743.

[11] H. Ren, S. Ruan, Y. Li, J. Bao, C. Meng, R. Li, and Y. Zheng, "MTrajRec: Map-constrained trajectory recovery via seq2seq multi-task learning," in *SIGKDD*, 2021, pp. 1410–1419.

[12] T. Xia, Y. Qi, J. Feng, F. Xu, F. Sun, D. Guo, and Y. Li, "AttnMove: History enhanced trajectory recovery via attentional network," in *AAAI*, vol. 35, no. 5, 2021, pp. 4494–4502.

[13] C. Yang and Z. Pei, "Long-short term spatio-temporal aggregation for trajectory prediction," *TITS*, vol. 24, no. 4, pp. 4114–4126, 2023.

[14] J. Wang, N. Wu, X. Lu, W. X. Zhao, and K. Feng, "Deep trajectory recovery with fine-grained calibration using Kalman filter," *TKDE*, vol. 33, no. 3, pp. 921–934, 2019.

[15] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. S. Quek, and H. Vincent Poor, "Federated learning with differential privacy: Algorithms and performance analysis," *IEEE T INF FOREN SEC*, vol. 15, pp. 3454–3469, 2020.

[16] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *AISTATS*, 2017, pp. 1273–1282.

[17] Z. Lai, D. Zhang, H. Li, C. S. Jensen, H. Lu, and Y. Zhao, "Lightcts: A lightweight framework for correlated time series forecasting," *SIGMOD*, vol. 1, no. 2, pp. 1–26, 2023.

[18] F. Chen, M. Luo, Z. Dong, Z. Li, and X. He, "Federated meta-learning with fast convergence and efficient communication," *arXiv preprint arXiv:1802.07876*, 2018.

[19] M. Delange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. Slabaugh, and T. Tuytelaars, "A continual learning survey: Defying forgetting in classification tasks," *TPAMI*, vol. 44, no. 7, pp. 3366–3385, 2022.

[20] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[21] Z. Liu, H. Miao, Y. Zhao, C. Liu, K. Zheng, and H. Li, "LightTR: A lightweight framework for federated trajectory recovery," in *ICDE*, 2024.

[22] Y. Chen, H. Zhang, W. Sun, and B. Zheng, "RNTrajRec: Road network enhanced trajectory recovery with spatial-temporal transformer," *arXiv:2211.13234*, 2022.

[23] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk, "On map-matching vehicle tracking data," in *PVLDB*, 2005, pp. 853–864.

[24] E. Rappos, S. Robert, and P. Cudré-Mauroux, "A force-directed approach for offline GPS trajectory map matching," in *SIGSPATIAL*, 2018, pp. 319–328.

[25] E. Chambers, B. T. Fasy, Y. Wang, and C. Wenk, "Map-matching using shortest paths," *TSAS*, vol. 6, no. 1, pp. 1–17, 2020.

[26] S. Zheng, Y. Yue, and P. Lucey, "Generating long-term trajectories using deep hierarchical networks," in *NIPS*, 2016, p. 1551–1559.

[27] Y. Lun, H. Miao, J. Shen, R. Wang, X. Wang, and S. Wang, "Resisting tul attack: balancing data privacy and utility on trajectory via collaborative adversarial learning," *GeoInformatica*, pp. 1–21, 2023.

[28] J. Fang, C. Zhu, P. Zhang, H. Yu, and J. Xue, "Heterogeneous trajectory forecasting via risk and scene graph learning," *TITS*, 2023.

[29] W. Chen, L. Chen, Y. Xie, W. Cao, Y. Gao, and X. Feng, "Multi-range attentive bicomponent graph convolutional network for traffic forecasting," in *AAAI*, vol. 34, no. 04, 2020, pp. 3529–3536.

[30] X. Wu, D. Zhang, C. Guo, C. He, B. Yang, and C. S. Jensen, "AutoCTS: Automated correlated time series forecasting," *PVLDB*, vol. 15, no. 4, pp. 971–983, 2021.

[31] X. Wu, D. Zhang, M. Zhang, C. Guo, B. Yang, and C. S. Jensen, "AutoCTS+: Joint neural architecture and hyperparameter search for correlated time series forecasting," *SIGMOD*, vol. 1, no. 1, pp. 1–26, 2023.

[32] J. Liang, X. Zhao, M. Li, Z. Zhang, W. Wang, H. Liu, and Z. Liu, "Mmmlp: Multi-modal multilayer perceptron for sequential recommendations," in *WWW*, 2023, pp. 1109–1117.

[33] M. Li, Z. Zhang, X. Zhao, W. Wang, M. Zhao, R. Wu, and R. Guo, "AutoMLP: Automated MLP for sequential recommendations," in *WWW*, 2023, pp. 1190–1198.

[34] S. J. Pan and Q. Yang, "A survey on transfer learning," *TKDE*, vol. 22, no. 10, pp. 1345–1359, 2009.

[35] S. Hoteit, S. Secci, S. Sobolevsky, C. Ratti, and G. Pujolle, "Estimating human trajectories and hotspots through mobile phone data," *Comput. Netw.*, vol. 64, pp. 296–307, 2014.

[36] J. Feng, C. Rong, F. Sun, D. Guo, and Y. Li, "PMF: A privacy-preserving human mobility prediction framework via federated learning," *IMWUT*, vol. 4, no. 1, pp. 1–21, 2020.

[37] Y. Zhang, L. Deng, Y. Zhao, J. Chen, J. Xie, and K. Zheng, "Simidtr: Deep trajectory recovery with enhanced trajectory similarity," in *DASFAA*, 2023, pp. 431–447.

[38] X. Luo, Y. Wu, X. Xiao, and B. C. Ooi, "Feature inference attack on model predictions in vertical federated learning," in *ICDE*, 2021, pp. 181–192.

[39] Y. Liu, Y. Kang, C. Xing, T. Chen, and Q. Yang, "A secure federated transfer learning framework," *IEEE INTELL SYST*, vol. 35, no. 4, pp. 70–82, 2020.

[40] L. Gao, H. Fu, L. Li, Y. Chen, M. Xu, and C.-Z. Xu, "Feddc: Federated learning with non-iid data via local drift decoupling and correction," in *CVPR*, 2022, pp. 10 112–10 121.

[41] E. Diao, J. Ding, and V. Tarokh, "HeteroFL: Computation and communication efficient federated learning for heterogeneous clients," *arXiv:2010.01264*, 2020.

[42] Y. Shi, Y. Tong, Y. Zeng, Z. Zhou, B. Ding, and L. Chen, "Efficient approximate range aggregation over large-scale spatial data federation," *TKDE*, vol. 35, no. 1, pp. 418–430, 2021.

[43] Y. Liu, J. J. Q. Yu, J. Kang, D. Niyato, and S. Zhang, "Privacy-preserving traffic flow prediction: A federated learning approach," *IoTJ*, vol. 7, no. 8, pp. 7751–7763, 2020.

[44] F. Yin, Z. Lin, Q. Kong, Y. Xu, D. Li, S. Theodoridis, and S. R. Cui, "Fedloc: Federated learning framework for data-driven cooperative localization and location data processing," *IEEE Open Journal of Signal Processing*, vol. 1, pp. 187–215, 2020.

[45] S. Thrun, "Is learning the n-th thing any easier than learning the first?" in *NeurIPS*, 1995, pp. 640–646.

[46] R. Aljundi, E. Belilovsky, T. Tuytelaars, L. Charlin, M. Caccia, M. Lin, and L. Page-Caccia, "Online continual learning with maximal interfered retrieval," in *NeurIPS*, 2019, pp. 11 849–11 860.

[47] J. Serra, D. Suris, M. Miron, and A. Karatzoglou, "Overcoming catastrophic forgetting with hard attention to the task," in *ICML*, 2018, pp. 4548–4557.

[48] J. Zhang, C. Chen, W. Zhuang, and L. Lyu, "Target: Federated class-continual learning via exemplar-free distillation," in *ICCV*, 2023, pp. 4782–4793.

[49] Y. Ma, Z. Xie, J. Wang, K. Chen, L. Shou, and L. De Raedt, "Continual federated learning based on knowledge distillation." in *IJCAI*, 2022, pp. 2182–2188.

[50] T. D. Bui, C. V. Nguyen, S. Swaroop, and R. E. Turner, "Partitioned variational inference: A unified framework encompassing federated and continual learning," *arXiv preprint arXiv:1811.11206*, 2018.

[51] J. Dong, L. Wang, Z. Fang, G. Sun, S. Xu, X. Wang, and Q. Zhu, "Federated class-incremental learning," in *CVPR*, 2022, pp. 10 164–10 173.

[52] D. Li, N. Huang, Z. Wang, and H. Yang, "Personalized federated continual learning for task-incremental biometrics," *IoTJ*, 2023.

Paper D

Federated Trajectory Similarity Learning with Privacy-Preserving Clustering

Hao Miao, Ziqiao Liu, Yan Zhao, Kai Zheng, Yupu Zhang,
Christian S. Jensen

The paper has been submitted to the
41th IEEE International Conference on Data Engineering, 2025.

© 201X IEEE
The layout has been revised.

Movement trajectory similarity computation is important when supporting functionalities such as outlier detection and prediction that may, in turn, fuel a variety of transportation-related applications. Recent trajectory similarity learning solutions often assume that trajectories are available at a central location. Yet, we are witnessing the decentralized collection of increasingly massive volumes of trajectories due to the deployment of edge devices. To enable decentralized training and improved privacy, we propose a federated trajectory similarity learning framework that features privacy-preserving clustering based on a client-server architecture. The framework encompasses local, client-side trajectory preprocessing and representation learning. This is combined with a novel privacy-preserving clustering mechanism that ensures consistent model updates between clients and the server, thus alleviating the effects of trajectory heterogeneity across clients. In addition, the framework features a hierarchical central aggregation mechanism that supports clustered federated learning. Extensive experiments on real data offer evidence that the effectiveness of the proposed framework performs as intended.

1 Introduction

With the proliferation of edge computing and devices enabling the capture of the movements of objects (e.g., vehicles and people), massive volumes of trajectory data are being collected in a decentralized fashion. This data has the potential to fuel a variety of trajectory applications [1, 2], such as traffic prediction [3], outlier detection [4], and Point-of-interest (POI) recommendation [1]. A fundamental aspect of such applications is trajectory similarity learning. The left part of Figure D.1 illustrates trajectory similarity, where we consider T_2 and T_3 as similar trajectories due to their similar characteristics (e.g., origin, destination, and timestamp t_i). In addition, the distance (e.g., Euclidean distance) between T_2 and T_3 is smaller than that between T_2 (or T_3) and T_1.

Due to its significance, substantial research has been devoted to inventing effective trajectory similarity models [2, 5–8]. Traditionally, studies measure trajectory similarity as the distance between trajectories, and both free-space measures [5, 9] and road network measures [10, 11] have been proposed. Recent studies [7, 12, 13] apply neural networks to compute the similarity between trajectories by learning deep representations of the trajectories. In real-world scenarios, trajectories are often generated at edge devices that are distributed across cities. Yet, most existing trajectory similarity learning models [7, 14] assume centralized training data and fail to take into account the decentralized setting. In addition to potentially better *scalability* by enabling more efficient processing and lower latencies, decentralized processing capabilities can enable improved data privacy [15–18] and can support data access restrictions

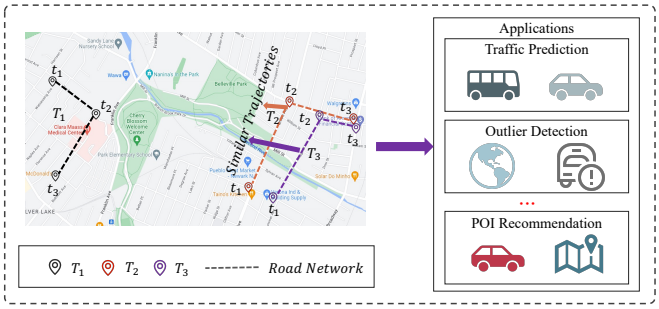


Fig. D.1: Illustration of Trajectory Similarity

due to existing licensing agreements [19]. There are thus good reasons to invent a new decentralized model that can utilize decentralized data and can learn complex spatio-temporal semantics of trajectories. However, it is non-trivial to develop such a model, due to two main challenges.

Challenge I: decentralized trajectory privacy protection. In order to achieve effective trajectory learning, existing methods require aggregation of all available trajectories, which may reveal real sensitive information that may be exploited by a malicious third party. With the increasing data privacy, such centralized processing may reduce the willingness of people to share their trajectory data, thus reducing the data available. Although recent studies in Federated Learning (FL) provide solutions for decentralized data training on multiple devices and considering privacy, these techniques cannot be applied to trajectory similarity learning directly due to the unique spatio-temporal characteristics of trajectories. To our knowledge, no FL models exist that can capture spatial dependencies and temporal correlations effectively in terms of trajectory similarity computation.

Challenge II: trajectory heterogeneity. The distributions of the trajectories that are collected across different edge devices are usually not independent and identically distributed (Non-IID) rather, they have been found to be remarkably heterogeneous [20, 21]. For example, the distributions of trajectories across children versus adults are different. Children may often be found in schools and playgrounds, while adults are more likely to be found in office buildings and shopping malls. When a decentralized model is trained with such heterogeneous local data, the model is unlikely to generalize well to other local datasets. The distribution drift across different local datasets makes it difficult to achieve an optimal global model by aggregating local models. It is highly desirable, but also non-trivial, to achieve a global model that is able to contend with data heterogeneity.

To address these challenges, we propose F-TSL, a federated trajectory similarity learning framework. F-TSL is designed to enable horizontal federated learning (i.e., server-clients), where many clients collaboratively train a global

model using local training data under the orchestration of a central server. F-TSL encompasses three major modules: local trajectory preprocessing and learning, privacy-preserving clustering, and server-side aggregation. By distributing the model training process across multiple clients, F-TSL exhibits scalability naturally. Especially, we can achieve an effective global model by selecting a subset of clients for global parameter aggregation in each training round.

To enable federated learning that supports the unique properties of trajectories, notably spatio-temporal correlations, we design a local trajectory preprocessing and learning module with a well-customized local trajectory learning model for use by clients. Specifically, a client first downloads the road network and parameters from the central server and then maps its trajectories onto the road network. Finally, the client utilizes the local trajectory learning model, which includes a spatio-temporal encoder and a spatio-temporal decoder, to learn effective spatio-temporal embeddings.

Next, the privacy-preserving clustering module aggregates similar clients together to contend with trajectory heterogeneity by means of a differential privacy method (i.e., Laplacian [22]) to avoid information leakage during clustering. This allows clients with similar trajectory data to profit from one another while reducing negative effects between clients with dissimilar data, thus facilitating more consistent model updates between the server and clients and improving the effectiveness of trajectory similarity learning. Considering the fine-grained information in trajectories, we employ the Hausdorff distance to compute trajectory similarity. In the server-side aggregation module, an additional alignment loss inspired by the Moreau envelope mechanism [16] is adopted for regularization in the training of clients to minimize the differences between local models trained by clients and the global model trained by the server.

The major contributions are summarized as follows.

- To the best of our knowledge, this is the first systematic study of federated trajectory similarity learning. We propose F-TSL, a federated similarity learning framework for decentralized trajectory similarity learning, which offers privacy protection of locally collected trajectories.
- We introduce a local trajectory preprocessing and learning module to capture effectively spatio-temporal correlations of trajectories by means of a well-customized local trajectory learning model.
- To contend with trajectory heterogeneity, a novel privacy-agnostic clustering mechanism is proposed to facilitate more consistent model updates between clients and the server. In addition, we introduce an alignment loss to minimize the differences between local models and the global model.

Table D.1: Summary of Notations

Symbol	Definition
G	Road network
V	Vertex set
E	Edge set
v_i	Road segment intersection or road end
$e_{i,j}$	Road segment from v_i to v_j
$\widetilde{p}_i$	GPS Point
τ	Raw trajectory
t_i	Timestamp of p_i
p_i	Map-matched point
T	Map-matched trajectory
Λ	Control parameter
$\mathcal{T}$	Map-matched trajectory dataset
$\mathcal{S}$	Central server

- We report on experiments on real data, offering evidence of the effectiveness of the new framework and its components.

The remainder of this paper is organized as follows, and Section 2 introduces preliminary concepts and the federated trajectory similarity problem. Section 3 analyzes the drawbacks of existing trajectory learning methods. We then present our solutions in Section 4, followed by the experimental evaluation in Section 5. Section 6 surveys the related work and Section 7 concludes the paper.

2 Preliminaries

We proceed to present necessary preliminaries and then define the problem addressed. Table D.1 lists the notations used throughout the paper.

Definition 32 (Road network) *A road network is modeled as a directed graph $G = (V, E)$, where V is a road vertex set, and E is an edge set. Each vertex $v_i \in V$ represents a road intersection or a road end, and each edge $e_{i,j} \in E$ indicates a directed road segment from v_i to v_j.*

Definition 33 (Point) *A point is a triple $p = \langle lat, lon, \gamma \rangle$ that records a latitude lat and longitude lon including additional information γ, e.g., address.*

Definition 34 (Raw Trajectory) *A trajectory τ is a sequence of points at n discrete timestamps, i.e.,*

$$\tau = \langle (\widetilde{p}_1, t_1), (\widetilde{p}_2, t_2), \cdots, (\widetilde{p}_n, t_n) \rangle, \tag{D.1}$$

where $\widetilde{p}_i$ denotes a geo-location (point) and t_i is the corresponding timestamp.

In practice, collected trajectory points may not be accurate due to GPS device measurement noises or errors. Map-matching is usually adopted to

project raw points onto an underlying road network [23]. We use the Incremental Map-Matching [24] method to convert a sequence of raw coordinates to coordinates in the road network that then become a sequence of road-network vertices that encode a connected sequence of edges. Specifically, the trajectory points are assumed to be located on the vertices in G. If a point $\widetilde{p}_i$ is located on an edge e, we split e into two sub-edges by introducing a new vertex p_i. Consequently, a raw trajectory τ is transformed into a directed path in G from a start vertex to an end vertex as follows.

Definition 35 (Map-matched Trajectory) *Given a raw trajectory τ and an underlying road network $G = (V, E)$, the trajectory T obtained by map-matching of τ is a sequence of n vertices in G, i.e.,*

$$T = \langle (p_1, t_1), (p_2, t_2), \cdots, (p_n, t_n) \rangle, \tag{D.2}$$

where $p_i \in V$ denotes a vertex in G and t_i represents the corresponding timestamp.

Definition 36 (Trajectory Similarity) *Given two trajectories T_i and T_j, the spatio-temporal similarity $\mathcal{D}(T_i, T_j)$ is defined as a weighted, linear combination of spatial and temporal similarity.*

$$\mathcal{D}(T_i, T_j) = \Lambda \cdot \mathcal{D}_{Spa}(T_i, T_j) + (1 - \Lambda) \cdot \mathcal{D}_{Tem}(T_i, T_j), \tag{D.3}$$

where $\mathcal{D}_{Spa}$ and $\mathcal{D}_{Tem}$ denote spatial and temporal similarity, respectively, and $\Lambda \in [0, 1]$ is a parameter controlling the contributions of spatial and temporal similarity, thus providing flexibility that can be used to support different applications as discussed in Section 1.

To define $\mathcal{D}$, we adopt an existing network-aware distance measure such as Two-Phase divide-and-conquer algorithm (TP) [25], Distributed In-Memory Trajectory Analytics (DITA) [10], Longest Common Road Segment (LCRS) [11]), and Network-aware Edit Distance with Real Penalty (NetERP) [14]. These distance measures can be used for spatial similarity computation, i.e., $\mathcal{D}_{Spa}$, as well as temporal similarity, i.e., $\mathcal{D}_{Tem}$, [25]. Please refer to the literature [26] for a more detailed coverage of these measures.

Based on the above definitions, we formally define the federated trajectory similarity learning problem as follows.

Federated Trajectory Similarity Learning. Consider a user with personal trajectories as a client. Given a central server $\mathcal{S}$ with a road network G, and $\mathcal{K}$ clients with local trajectory datasets $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \cdots, \mathcal{T}_\mathcal{K}\}$, the problem is to learn a shared global score function $\mathcal{F}(\cdot, \cdot)$, such that for any pair of trajectories T_i and T_j from τ in each client, the difference $|\mathcal{F}(T_i, T_j) - \mathcal{D}(T_i, T_j)|$ is minimized. Note that each client keeps its local data and only transfers parameters to the central server during training by considering privacy.

3 Discussion

We proceed to first analyze existing trajectory similarity computation proposals in a generic framework and then identify directions for achieving a decentralized trajectory similarity learning framework.

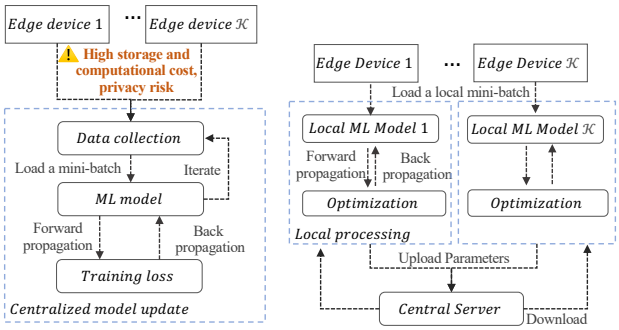


Fig. D.2: (a) A generic framework of existing trajectory recovery methods. (b) An illustration of the federated learning framework.

To understand the prospects of a decentralized trajectory similarity learning method, we consider existing representative trajectory similarity learning methods [7, 12, 13, 23], that perform in a centralized fashion. A generic framework for these methods is shown in Figure D.2. Generally, a trajectory similarity learning encompasses three components:

1. *a data collection component* that collects data from edge devices to a central server;
2. *a machine learning (ML) model* consisting of different spatio-temporal neural networks to extract spatio-temporal correlations from historical trajectories as high-dimensional features;
3. *an optimization component* that updates the model iteratively based on mini-batches of data by means of forward and back propagation.

It is obvious that existing trajectory similarity learning methods perform in a centralized fashion while are unable to contend with a setting where trajectories are not stored at a central server. Although the models learned on centralized trajectories by these methods bring the benefits of greatly improving the usability of real-world applications, there are also additional risks and responsibilities associated with storing trajectories in a centralized location. With strict privacy regulations in place and given the sensitive nature and monetary value of trajectory data, data owners not not be willing to share their raw trajectories to a third party.

This motivates us to design a decentralized trajectory similarity learning model, where data and computation are distributed among multiple edge devices, which enables improved data privacy protection, potentially more efficient processing and lower latencies. We thus propose a new federated trajectory similarity learning framework based on horizontal federated learning [27], to be detailed in Section 4. Figure D.2(b) shows the typical learning process of a horizontal federated learning framework. Specifically, each client first learns a local model using its local data, while the central server aggregates the parameters of the local models periodically to update its global model.

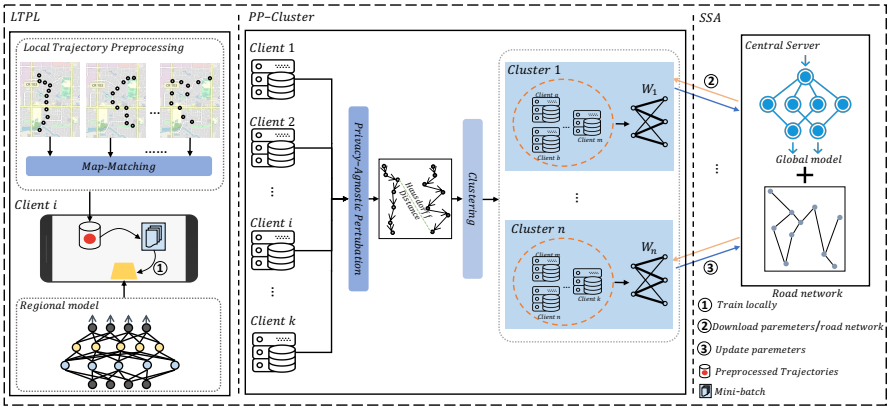


Fig. D.3: F-STL Framework Overview

The novelties of the proposed framework are as follows. First, we first study the federated trajectory similarity learning systematically. Second, we design a well-customized local trajectory learning model to consider the unique spatio-temporal correlations and the road network information. Third, we design a novel privacy-preserving clustering mechanism to contend with the data heterogeneity across clients by considering unique trajectory characteristics.

As described in many existing studies [28–31], Federated learning is a machine learning setting ensuring privacy protection. We aim to enable trajectory similarity learning across different clients (i.e., edge devices) without modifying the implementations of the general horizontal federated learning framework (i.e., a client-server architecture) [28, 31], that guarantees privacy by asking clients to share local model parameters instead of their local trajectory data. In addition, client updates are encrypted to ensure that there is no privacy leakage of clients at the central server following the study [31]. We do not include a formal coverage of privacy protection, but instead, refer interested readers to the literature [28, 29, 31]. The proposed F-TSL offers two benefits related to privacy. First, F-TSL alleviates sensitive information leakage, since it keeps the training data decentralized and only sends model parameters to the central server. Second, F-TSL is robust to potential attacks. An attacker

may need to compromise multiple clients simultaneously to gather a meaningful amount of sensitive data. By distributing the learning across multiple clients, F-TSL mitigates malicious attacks, making it harder for an untrusted third party to use the real trajectories.

4 Methodology

We detail the Federated Trajectory Similarity Learning (F-TSL) framework for decentralized trajectory similarity computation. We first give an overview of the framework and then provide specifics on each module in the framework.

4.1 Framework Overview

F-TSL is based on horizontal learning and uses a client-server structure to collaboratively train local trajectory learning models under the orchestration of a central server, as illustrated in Figure D.3. It consists of three major modules: local trajectory preprocessing and learning, privacy-preserving clustering, and server-side aggregation.

- *Local trajectory preprocessing and learning (LTPL).* In this module, each client first preprocesses locally collected trajectory data. Meanwhile, it downloads parameters and the global road network and then maps all trajectories to the corresponding roads. Finally, the client learns a local model to quantify local trajectory similarity.
- *Privacy-Preserving Clustering (PP-Cluster).* Clients with different trajectories often exhibit remarkable heterogeneity, without the consideration of which performance will degrade. In this module, a novel privacy-preserving clustering mechanism aggregates similar clients by means of Hausdorff distance for consistent model updates between clients and the server.
- *Server-Side Aggregation (SSA).* In this module, each cluster aggregates parameters from its clients to obtain a regional model. The central server aggregates these regional models based on the road network to update the global similarity score function.

Next, we present each module in detail.

4.2 Local Trajectory Preprocessing and Learning

In this section, we cover the preprocessing and learning of local trajectories in a client. Note that the model for the central server and the models for all clients

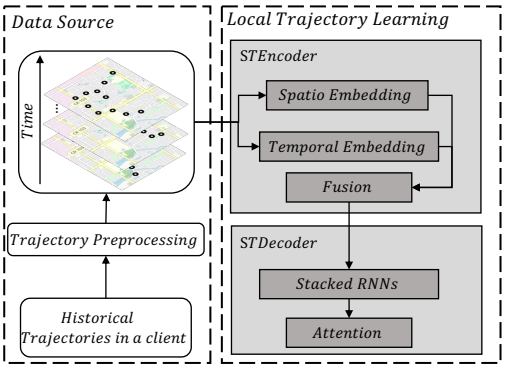


Fig. D.4: Local Trajectory Learning Model

share the same trajectory learning model structure, as shown in Figure D.4. We elaborate the trajectory learning model in Section 4.2.

Local Trajectory Preprocessing

Given a road network that is downloaded from the central server, we first map all trajectories to the corresponding road intersections according to the Incremental Map-Matching (IMM) method [24]. This way, we convert the raw GPS data into time-ordered trajectories as follows.

$$T = \langle (p_1, t_1,), (p_2, t_2), \cdots, (p_n, t_n) \rangle, \tag{D.4}$$

where p_i is i-th point at time stamp t_i and n denotes the trajectory length. To tackle the issue of noisy data and to learn powerful trajectory representations, we remove trajectories with fewer than 10 points (i.e., $n \leq 10$). Further, to facilitate the capture of spatial and temporal correlations, we further separate T into $T_s = \langle p_1, p_2, \cdots, p_n \rangle$ and $T_t = \langle t_1, t_2, \cdots, t_n \rangle$ that represent the spatial and temporal aspects of T, respectively.

Local Trajectory Learning

Existing trajectory embedding methods focus predominantly on capturing spatial correlations of trajectories while ignoring temporal dependencies. To better capture spatial and temporal dependencies of trajectories, inspired by [23], we introduce a local trajectory learning method, including a spatio-temporal encoder and a spatio-temporal decoder, which uses each client's local data to learn effective trajectory embeddings.

Spatio-Temporal (ST) Encoder. Given T_s and T_t, the ST encoder (see Figure D.4) aims to learn spatial and temporal embeddings of trajectories. It first learns spatial and temporal embeddings and then combines them.

$$
\begin{aligned}
h_s &= SpatialEmbedding(T_s) \\
h_t &= TemporalEmbedding(T_t) \\
h &= h_s + h_t,
\end{aligned} \tag{D.5}
$$

where h_s and h_t denote the learned spatial and temporal embeddings, respectively, + is element-wise addition.

Spatial Embeddings. Spatial embeddings are designed to learn spatial correlations of trajectories. Given T_s, we first adopt Node2Vec [32] to capture the co-occurrence of the adjacent locations, which aims to learn road structure information. Specifically, given a point p_i, we obtain its embedding e_i by approximating the spatial conditional probability of its nearby points as follows.

$$
e_i = Node2Vec(p_i) \tag{D.6}
$$

After obtaining all the embeddings $emb = \langle e_1, e_2, \cdots, e_n \rangle$ of T_s, we input emb into stacked GCN layers to learn final spatial embeddings.

$$
h_{p_i} = \sigma((e_i \oplus \sum_{j \in N_i} a_{i,j} W_s e_j) + b_{p_i}), \tag{D.7}
$$

where $\oplus$ denotes the concatenation operation, N_i is the neighborhood set of p_i, $a_{i,j}$ is an adjacency weight, W_s is a learnable parameter, and b_{p_i} represents the bias. Based on Equations D.6 and D.7, we learn high-dimensional features, $h_s = \langle h_{p_1}, h_{p_2}, \cdots, h_{p_n} \rangle$, which captures spatial correlations of T_s.

Temporal Embeddings. Given T_t, we embed it to get a vector with size of $l+1$ for each timestamp t as follows.

$$
s_t[i] = \begin{cases} w_i t + b_i, & i = 0 \\ cos(w_i t + b_i), & 1 \leq i \leq l, \end{cases} \tag{D.8}
$$

where w_i and b_i represent learnable parameters and bias, respectively, and $cos(\cdot)$ is the cosine function. Based on Equation D.8, we embed T_t into time vectors $h_t^s = \langle s_1, s_2, \cdots, s_n \rangle$. Then we input h_t^s into stacked recurrent neural networks (RNNs) to further learn high-dimensional temporal dependencies of T_t.

$$
h_t = RNNs(h_t^s) \tag{D.9}
$$

Spatio-Temporal (ST) Decoder. The learned spatio-temporal embeddings h (see Equation D.5) are then input into the ST decoder to decode the data representations for similarity computation. As shown in Figure D.4, the ST decoder contains several stacked RNNs, $RNNs(\cdot)$, followed by an attention

layer, $Attention(\cdot)$, to learn a projection function for computing similarities. It can be formulated as follows.

$$\begin{aligned} h_{decode} &= RNNs(h) \\ v &= Attention(h_{decode}) = \sum_{i=0}^{n} \alpha_i h_{decode}[i], \end{aligned} \tag{D.10}$$

where $\alpha_i = \frac{exp(h_{decode}[i])}{\sum_{j=0}^{n} exp(h_{decode}[j])}$, and v is the learned embeddings.

Following previous studies [7, 23], the dot product is adopted to measure the similarity between embeddings of trajectories. Specifically, for a pair of trajectories T_i and T_j, we learn their respective embeddings v_{T_i} and v_{T_j}, and the similarity score $\mathcal{F}(T_i, T_j) = v_{T_i}^{\mathsf{T}} v_{T_j}$, where T is the transpose operation.

4.3 Privacy-Preserving Clustering

To alleviate the problem of data heterogeneity, we propose a novel privacy-preserving clustering mechanism to aggregate similar clients together by calculating the Hausdorff distance between locally perturbed embeddings of clients, aiming at achieving more focused model updates to ensure consistent model updates between the server and clients. Considering privacy protection, we first perturb the locally learned trajectory embeddings $V = \langle v_1, v_2, \cdots, v_l \rangle$ of each client to $V' = \langle v_1', v_2', \cdots, v_l' \rangle$ by means of differential privacy, where l is the length of the local dataset. Then, we partition all clients into clusters $\mathcal{M} = \{M_1, M_2, \cdots, M_{\mathcal{M}}\}$, where $\sum_{i=1}^{m} |M_i| = \mathcal{K}$, based on the Hausdorff distance between the perturbed embeddings of clients, which considers the information of individual trajectories.

We first give two necessary definitions: ϵ-differential privacy [33] and sensitivity.

Definition 3 (ϵ-differential privacy) *A random algorithm M is ϵ-differentially private if $\forall S \subseteq OutRange(M)$ and two datasets X and $Y = [X; z](z \in \mathbb{R}^d)$, we have*

$$Pr[M(x) \in S] \leq e^{\epsilon} Pr[M(Y) \in S]. \tag{D.11}$$

Definition 4 (sensitivity) *For a function f, the sensitivity of f is defined as follows.*

$$\Delta f = \max_{X,Y} ||f(X) - f(Y)||_1 \tag{D.12}$$

Privacy-Agnostic Perturbation. Using local data from clients directly when performing clustering may cause sensitive information leakage. To avoid this problem, we propose a privacy-agnostic perturbation method based on differential privacy (i.e., a Laplacian mechanism). In particular, given a client c_i with

local trajectories $\mathcal{T}_i$ and local trajectory learning model $f(\cdot)$, we perturb the original trajectories by adding Laplacian noise to their learned high-dimensional embeddings V.

$$\begin{aligned} &M(\mathcal{T}_i) = f(\mathcal{T}_i) + z = V + z, \\ &z_i \sim Lap(0, \frac{\Delta f}{\epsilon}), z_i \in z \\ &V' = M(\mathcal{T}_i), \end{aligned} \tag{D.13}$$

where $z = (z_1, z_2, \cdots, z_d)$, d is the feature dimension of V, and $Lap(\cdot)$ is the Laplacian distribution.

Theorem 2 *$V' = M(\mathcal{T}_i)$ preserves $d\epsilon$-differential privacy.*

proof 2 *Let p_x and p_y denote the probability density of $M(x)$ and $M(y)$ at an arbitrary output v', respectively. We have:*

$$\begin{aligned} \frac{p_x(v')}{p_y(v')} &= \prod_{i=1}^{q} \left(\frac{\exp\left(-\frac{d\epsilon \left| f(x)_i - z_i \right|}{\Delta f} \right)}{\exp\left(-\frac{d\epsilon \left| f(y)_i - z_i \right|}{\Delta f} \right)} \right) \\ &= \prod_{i=1}^{q} \exp\left(\frac{d\epsilon \left(\left| f(y)_i - z_i \right| - \left| f(x)_i - z_i \right| \right)}{\Delta f} \right) \\ &\leq \prod_{i=1}^{q} \exp\left(\frac{d\epsilon \left| f(x)_i - f(y)_i \right|}{\Delta f} \right) \\ &= \exp\left(\frac{d\epsilon \cdot \| f(x) - f(y) \|_1}{\Delta f} \right) \leq \exp(d\epsilon), \end{aligned} \tag{D.14}$$

where x and y are two trajectories in $\mathcal{T}_i$, and q is the number of the feature dimensions.

We use the Laplacian mechanism to perturb embeddings for all clients. Specifically, for client c_i, we perturb embeddings $V' = \langle v'_1, v'_2, \cdots, v'_l \rangle$ for the following Hausdorff distance based clustering.

Hausdorff Distance based Clustering. Most existing federated clustering methods [34, 35] consider a client as a whole and perform clustering at the client level, ignoring the influence of fine-grained local data (i.e., client-specific trajectories), which hurts trajectory learning performance [36]. To solve this problem, we design a Hausdorff distance based clustering mechanism to aggregate similar clients in high-dimensional feature spaces, by measuring the maximum mismatch degree among trajectory embedding sets in clients.

The Hausdorff distance based clustering is shown in Algorithm 12. Specifically, we first select $\mathcal{M}$ cluster centers that are far away from each other in the

Algorithm 12 The Hausdorff Distance based Clustering

Input: a set of clients $\mathbf{C} = \{c_1, \cdots, c_\mathcal{K}\}$ with locally perturbed trajectory embeddings $V_i' \subseteq c_i$, the number of cluster centers $\mathcal{M}$
Output: $\mathcal{M}$ cluster centers: $M_1, M_2, \cdots, M_\mathcal{M}$

1: Randomly select a client c_i in $\mathbf{C}$ as the first cluster center M_1
2: $\overline{\mathbf{C}} \leftarrow C - c_i$
3: **for** c_j in $\overline{C}$ **do**
4: Compute Hausdorff distance between embeddings of client $c_j (V_j' \subseteq c_j)$ and cluster $M_1 (V_i' \subseteq c_i)$ by Equation D.15
5: $Dis(c_j) \leftarrow HD(V_j', V_i')$
6: **end for**
7: Sort $\overline{\mathbf{C}}$ by $Dis(\cdot)$ in descending order
8: Select top $\mathcal{M} - 1$ clients $\mathbf{C}_{ini}$ in sorted $\overline{\mathbf{C}}$ as remaining initial cluster centers
9: $\overline{\mathbf{C}} \leftarrow \overline{\mathbf{C}} - \mathbf{C}_{ini}$
10: **while** $\mathcal{M}$ is changed **do**
11: **for** c_k $(V_k' \subseteq c_k)$ in $\overline{\mathbf{C}}$ **do**
12: **for** M_m $(V_m' \subseteq M_m)$ in $\mathcal{M}$ **do**
13: Compute Hausdorff distance between embeddings of client c_k and cluster center $M_\mathcal{M}$ by Equation D.15
14: $Dis_c(M_m) \leftarrow HD(V_k', V_m')$
15: **end for**
16: $m^* \leftarrow argmin_m Dis_c(M_m)$ % *Assign c_k to closest center*
17: $M_{m^*} \leftarrow M_{m^*} \cup c_k$
18: $V_{m^*}' \leftarrow V_{m^*}' \cup V_k'$
19: **end for**
20: **end while**

high-dimensional space (lines 1–9). Then, for unselected client c_j, we compute the Hausdorff distance among embeddings of client c_j and $\mathcal{M}$ cluster centers in the high-dimensional feature space (lines 12–15). The computation of Hausdorff distance is given by Equation D.15. Finally, we group the client with its nearest cluster center (lines 16–18) until cluster centers are unchanged.

Hausdorff Distance Computation. Given two clients with local perturbed embeddings, V_i' and V_j', we compute their Hausdorff distance $HD(\cdot)$ as follows.

$$\begin{aligned} HD(V_i', V_j') &= \max(m(V_i', V_j'), m(V_j', V_i')), \\ m(V_i', V_j') &= \max_{v_i' \in V_i'} (\min_{v_j' \in V_j'} (dist(v_i', v_j'))), \\ m(V_j', V_i') &= \max_{v_j' \in V_j'} (\min_{v_i' \in V_i'} (dist(v_j', v_i'))), \end{aligned} \tag{D.15}$$

where $dist(\cdot)$ is the Euclidean distance between high-dimensional perturbed trajectory embeddings.

Figure D.5 shows an example of Hausdorff distance computation between two local embedding sets $V_i' = \langle v_{i,1}', v_{i,2}' \rangle$ and $V_j' = \langle v_{j,1}', v_{j,2}', v_{j,3}' \rangle$. For em-

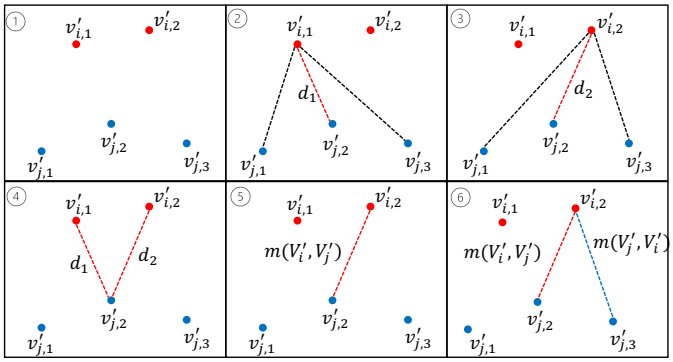


Fig. D.5: Hausdorff Distance Computation

bedding $v'_{i,1}$ in V'_i, we calculate the Euclidean distance between $v'_{i,1}$ and any embedding in V'_j iteratively, where d_1 is the minimum distance. Similarly, we find that d_2 is the minimum distance between $v_{i,2}$ and V'_j, where $d_2 > d_1$. Then, we use d_2 for $m(V'_i, V'_j)$, which is the shortest distance between V'_i and V'_j. Since we have a directed road network, we obtain the shortest distance $m(V'_j, V'_i)$ between V'_j and V'_i similarly. Finally, it is easy to see that $m(V'_i, V'_j)$ is smaller than $m(V'_j, V'_i)$. Thus we use $m(V'_j, V'_i)$ as the Hausdorff distance between V'_i and V'_j.

4.4 Server-Side Aggregation

We propose a server-side aggregation method to collaboratively train the clustered clients, which first aggregates parameter updates on a local cluster with a subset of clients and then performs parameter aggregation among clusters.

In federated trajectory similarity learning, we have $\mathcal{K}$ clients, denoted as $\{c_1, c_2, \cdots, c_{\mathcal{K}}\}$, each with its own dataset $\mathcal{T}_i$ that has three parts: a training, a validation, and a test dataset, denoted as $\mathcal{T}_i^{train}$, $\mathcal{T}_i^{valid}$, and $\mathcal{T}_i^{test}$, respectively. We aim to learn a good trajectory recovery model $\mathcal{F}_i(\cdot)$ for each client c_i on its local dataset $\mathcal{T}_i$:

$$min \frac{1}{N} \sum_{i=1}^{N} \frac{1}{n_i^{test}} \sum_{j=1}^{n_i^{test}} L_{local}(\mathcal{F}_i(T_j, T_n, T_q)), \tag{D.16}$$

where L_{local} is the local loss function, to be detailed in Equation D.17, n_i^{test} is the length of $\mathcal{T}_i^{test}$, and T_n and T_p are sampled similar and dissimilar trajectories of T_i, respectively.

Next, we introduce the local objective function of each client and hierarchical parameters updating in a federated training way.

Local Objective Function. The local objective function contains two parts, a loss of trajectory similarity L_{sim} and a client-cluster alignment loss L_{align},

Algorithm 13 Hierarchical Parameter Update

Input: R: communication rounds, θ_s^0: initial global model of server, M: cluster numbers, E: local training epochs, α: learning rate
Output: θ_s: final global model of server

for each round r in $[1, 2, \cdots, R]$ **do**
  Server transmits θ_s^{r-1} to M cluster centers
  $\Theta_m \leftarrow \theta_s^0$ for cluster M_m in M clusters
  for M_m in M clusters **do**
    Sample F fraction of clients in M_m
    Transmit Θ_m to selected clients in M_m
    for each selected client $c_i \in M_m$ **do**
      for each local epoch e in $[1, \cdots, E]$ **do**
        $\theta_{c_i}^r \leftarrow arg\min_\theta L_{local}(\theta_{c_i}^r)$
        $\nabla L_{local}(\theta_{c_i}^r) \leftarrow \nabla L_{local}(\theta_{c_i}^{r-1}) - \alpha(\theta_{c_i}^r - \theta_{c_i}^{r-1})$
      end for
      $\theta_{c_i} \leftarrow \theta_{c_i}^r$
      Transmit local client model θ_{c_i} to the respective cluster
    end for
    $\Theta_m \leftarrow \sum_{i=1}^{|M_m|} \frac{|c_i|}{|M_m|} \theta_{c_i}$
  end for
  $\theta_s \leftarrow \sum_{m=1}^{M} \frac{1}{M} \Theta_m$
end for

for alleviating data heterogeneity. We combine them together, and the overall local loss of a client is as follows.

$$L_{local} = L_{sim} + L_{align} \tag{D.17}$$

Following existing studies [7, 23] on trajectory similarity, we adopt a triplet-based pair-wise loss for effective and robust training.

$$L_{sim} = \max \sum_{T_m \in \mathcal{T}} \sum_{T_n \in \mathcal{T}, T_q \in \mathcal{T} \setminus \{T_m\}} \mathbb{I}(\mathcal{F}(T_m, Tn) < \mathcal{F}(T_m, T_q)), \tag{D.18}$$

where T_m is a randomly selected trajectory in $\mathcal{T}$, T_n and T_p are sampled similar and dissimilar trajectories of T_m, respectively, and $\mathbb{I}(\cdot)$ is an indicator function, where it equals one if the condition (i.e., $\mathcal{F}(T_m, Tn) < \mathcal{F}(T_m, T_q)$) holds, otherwise it is zero. Note that we randomly sample the trajectory of T_m instead of traversing all trajectories to improve efficiency.

To contend with data heterogeneity, inspired by the Moreau envelopes in personalized federated learning [37], we propose a simple but effective client-cluster alignment loss to balance the updates between clients and the cluster center. For a client c_i in cluster M_j, we have

$$L_{align} = \frac{\lambda}{2} ||\theta_i - \Theta_j||^2, \tag{D.19}$$

Table D.2: *Statistics of Datasets*

Dataset	*Tdrive*	*TaxiRome*
City name	Beijing	Rome
Time span	2/2/2008 ~ 8/2/2008	1/2/2014 ~ 2/3/2014
Number of taxis	10,357	320
Number of points	15 million	367,052
Collection type	GPS	GPS

where λ is a control weight, θ_i is the local model of client c_i, and Θ_j is an aggregated model of cluster j. We combine L_{sim} and L_{align} to train a suitable local model for a client and ensure it not drift off the aggregated model of its corresponding cluster.

Hierarchical Parameter Updating. We propose a hierarchical parameter updating method to perform clustered federated training, as shown in Algorithm 13. In particular, at each communication round r, the central server transmits its current model θ_s^{r-1} to each cluster (lines 1–3). Each cluster transmits its current parameters Θ_m to a subset (F fraction) of the clients, aiming to reduce the training time (lines 4–6). Each active client optimizes a local empirical risk objective, which is the sum of similarity loss and alignment loss (lines 7–11). Next, the cluster updates its universal model (lines 12–15) to proceed with more focused model updates, i.e., the model updates of clients are consistent with those of the server. Finally, the central server aggregates parameters from clusters to update its model θ_s until it converges to a stationary point θ_s (line 17).

5 Experimental Evaluation

5.1 Experimental Setup

Datasets

The experiments are carried out on two widely-used public trajectory datasets: Tdrive and TaxiRome.

- **Tdrive**[1]**.** The Tdrive dataset contains taxi-trip-based trajectories including 10357 taxi drivers with 15 million GPS points. It covers taxi trips in Beijing from Feb. 2 to 8, 2008.
- **TaxiRome**[2]**.** The TaxiRome dataset contains 367,052 taxi GPS points generated by 320 taxi drivers, which are collected in Rome over 30 days.

The dataset statistics are provided in Table D.2, which includes the city name, the time span, the number of taxis, and the number of points.

[1]https://www.microsoft.com/en-us/research/publication/t-drive-trajectory-data-sample/
[2]https://crawdad.org/roma/taxi/20140717/

Baselines

We compare the proposed F-TSL with the following baseline methods.

- **FC+FL**. The FC+FL method employs stacked fully connected (FC) networks as the local model trained with FedAvg [28] under the horizontal federated learning (FL) framework.
- **RNN+FL**. The RNN+FL combines horizontal FL with a stacked RNN to train a decentralized model for trajectory similarity computation.
- **GCN+FL**. GCN+FL is a decentralized trajectory similarity computation model, which uses stacked graph convolutional network (GCN) integrated with horizontal FL to collaboratively learn trajectory embeddings.

Evaluation Metrics

Following existing learning-based trajectory similarity studies [7, 23], *top-k* similarity search is utilized for experimental validation. We use *HR*@10, *HR*@50, and *R10*@50 as evaluation metrics. Specifically, *HR@k* denotes the *top-k* hitting ratio that is the overlap degree between a *top-k* result and the corresponding ground truth. For each client, we have

$$HR@K = \frac{1}{\mathcal{T}^{te}} \sum_{T \in \mathcal{T}^{te}} \frac{|L_T^P@K \cap L_T^G|}{|L_T^G|}, \tag{D.20}$$

where $\mathcal{T}^{te}$ is the local test set of trajectories, $L_T^P@K$ is the list of the predicted trajectories that are most similar to a given trajectory T with length K, L_T^G is the set of similar trajectories in the testing set for the given trajectory T, and $|\cdot|$ is set cardinality. *Rk@t* is the *top-t* recall for the *top-k* ground truth that captures the fraction of the *top-k* ground truth in the corresponding top-t result. In addition, we adopt four traditional trajectory similarity measure methods to obtain the ground truth results of top-k similarity search, i.e., Two-Phase divide-and-conquer algorithm (**TP**) [25], Distributed In-Memory Trajectory Analytics (**DITA**) [10], Longest Common Road Segment (**LCRS**) [11]) and Network-aware Edit Distance with Real Penalty (**NetERP**) [14]. The larger the HR@10, the HR@50, and the R10@50 are, the more accurate the method is.

Complexity

The time complexity of the proposed F-TSL is $O(D^2 \cdot N \cdot L)$, and the space complexity is $O(D^2)$, where D is the number of embedding dimensions, N is the number of trajectories, and L is the maximum length of historical trajectories (historical trajectories may have different lengths).

Table D.3: Overall Accuracy on Two Datasets

Datasets	Baselines	TP			DITA			LCRS			NetERP		
		HR@10	HR@50	R10@50	HR@10	HR@50	R10@50	HR@10	HR@50	R10@50	HR@10	HR@50	R10@50
Tdrive	**FC+FL**	0.0087	0.0230	0.0279	0.0165	0.0351	0.0556	0.0059	0.0208	0.0267	0.0412	0.0739	0.1018
	RNN+FL	0.1120	0.1753	0.2928	**0.1476**	0.1507	0.1566	0.1028	0.1983	0.2214	0.1061	0.1162	**0.2077**
	GCN+FL	0.0078	0.0224	0.0320	0.0071	0.0246	0.0312	0.0106	0.0241	0.0354	0.0251	0.0621	0.0713
	F-TSL	**0.1769**	**0.2585**	**0.3709**	0.1366	**0.1670**	**0.2520**	**0.1330**	**0.2088**	**0.3531**	**0.1496**	**0.1745**	0.1816
TaxiRome	**FC+FL**	0.0204	0.0524	0.0752	0.0246	0.0593	0.0833	0.0165	0.0384	0.0567	0.0589	0.0110	0.1659
	RNN+FL	0.1059	0.1894	0.2690	0.0619	0.1241	0.1663	0.0582	0.0964	0.1683	0.0514	0.0889	0.1311
	GCN+FL	0.0068	0.0231	0.0278	0.0095	0.0324	0.0358	0.0068	0.0268	0.0314	0.0279	0.0803	0.0929
	F-TSL	**0.1663**	**0.2472**	**0.3396**	**0.1278**	**0.1881**	**0.2633**	**0.0698**	**0.1033**	**0.1749**	**0.0819**	**0.1256**	**0.1765**

Implementation Details

We implement our model with the Pytorch framework on NVIDIA GTX 1080Ti GPU. The parameters in the model are set as follows. We split the local dataset in each client into training, validation, and test sets with proportion $3:1:6$. We set Λ (that is control weight in Equation.D.3) as 0.5 as the default. We set batch size to 50 for mini-batch learning. The initial learning rate is 0.001. The hidden feature dimensions of the spatial and temporal embeddings are both set to 128. The RNN structure contains one layer with the number of hidden features being 128. We add Laplacian noise z_i to each element of V. In addition, the numbers of clients and clusters are set to 100 and 10 as default, respectively. In this study, we apply IMM [24] to map-match original trajectories for subsequent model performance comparison. The code and additional experiments can be found at `https://github.com/MiaoHaoSunny/F-TSL`.

5.2 Experimental Results

Overall Accuracy

Table D.3 shows the performance comparison among different methods on the two datasets. The best performance by an existing method (FC+FL, RNN+FL, and GCN+FL) is underlined, and the overall best performance is marked in bold.The observations are as follows.

- Our F-TSL achieves the best results among all the baselines in most cases on the two datasets. F-TSL performs better than the best among the baselines by up to 57.94% (HR@10), 60.92% (R10@50), 59.48% (R10@50), and 50.17% (HR@50) for TP, DITA, LCRS, and NetERP, respectively on Tdrive. We observe that the performance improvements obtained by F-TSL on Tdrive exceed those on TaxiRome, due to the fact that Tdrive has much more training data than TaxiRome. Thus, the methods trained with sufficient data get better results.
- RNN+FL, which can capture temporal dependencies of trajectories, performs much better than FC+FL and GCN+FL (that can capture spatial dependencies) on both Tdrive and TaxiRome. It shows that effectively capturing temporal dependencies of trajectories is more important than

Table D.4: Effect of the Number of Clients on Both Datasets

Datasets	Metrics		Number of Clients			
			20	50	80	100
Tdrive	TP	HR@10	0.0676	0.1792	**0.1831**	0.1769
		HR@50	0.1877	0.2311	0.2315	**0.2585**
		R10@50	0.2230	0.3157	0.3450	**0.3709**
	DITA	HR@10	0.0314	0.1235	**0.1596**	0.1366
		HR@50	0.0884	0.1584	**0.1805**	0.1670
		R10@50	0.0982	0.2023	**0.2666**	0.2520
	LCRS	HR@10	0.1154	0.1252	0.1160	**0.1330**
		HR@50	0.1708	0.1889	0.1499	**0.2088**
		R10@50	0.3161	0.3504	0.3356	**0.3531**
	NetERP	HR@10	0.0398	0.1216	0.1074	**0.1496**
		HR@50	0.0898	0.1321	0.1337	**0.1745**
		R10@50	0.1120	0.1547	**0.2022**	0.1816
TaxiRome	TP	HR@10	0.1097	0.1508	0.1353	**0.1663**
		HR@50	0.1601	0.2424	**0.2483**	0.2472
		R10@50	0.2010	0.3301	0.3308	**0.3396**
	DITA	HR@10	0.1085	0.1017	0.0812	**0.1278**
		HR@50	0.1245	0.1716	0.1475	**0.1881**
		R10@50	0.1946	0.2298	0.2093	**0.2633**
	LCRS	HR@10	0.0624	0.1039	10.1633	**0.1698**
		HR@50	0.0666	0.1113	**0.1562**	0.1033
		R10@50	0.1795	0.1953	0.2606	**0.2749**
	NetERP	HR@10	0.0538	0.0619	0.0569	**0.0819**
		HR@50	0.1157	0.1126	0.09707	**0.1256**
		R10@50	0.1117	0.1459	0.1455	**0.1765**

learning spatial correlations since the originally collected trajectories are preprocessed into sequence data.

- Compared with the state-of-the-art centralized method ST2Vec, F-TSL performs worse than ST2Vec over Tdrive, while performs comparably over TaxiRome since it is easier for the centralized method to learn commonalities from centralized data with shuffled mini-batch training. But the centralized methods hurt privacy severely, which makes the learned model vulnerable.

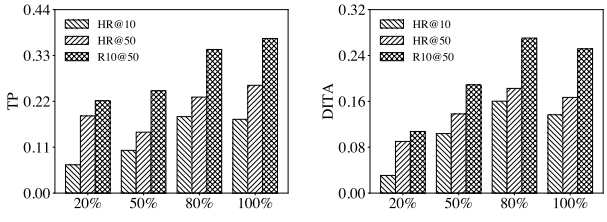
The above observations indicate that F-TSL is more effective than the existing FL-based methods.

Effect of the Number of Clients

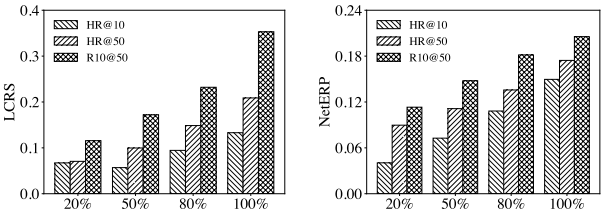
To study the effect of the number of clients participating in model training of our F-TSL, we conduct experiments with 20, 50, 80, and 100 clients. The results are reported in Table D.4. One can see that the model performance exhibits an increasing trend as the number of clients increases during training in most cases. For example, the HR@50 of TP on Tdrive increases from 0.1877 to 0.2585, and the HR@10 of LCRS on TaxiRome increases from 0.0624 to 0.1698. The results show that more clients participating in training is likely to result in better performance. This is because more clients with more local training data involved in global aggregation introduces more useful knowledge, which leads to higher accuracy.We also observe that the 100-client-based F-TSL performs worse than the 80-client-based F-TSL.

Effect of Client Fractions

When aggregating parameters in F-TSL, we sample a certain fraction of clients to perform model training in each cluster. To study the effect of client fraction F on the model performance, we conduct experiments by sampling four different fractions of clients: 20%, 50%, 80%, and 100%. The results are shown in Figure D.6. Generally, the model performance gets better with the increase of client fractions. The performance is improved by at least 39.65% evaluated by four different metrics (i.e., TP, DITA, LCRS, and NetERP) when the client fraction changes from 20% to 100%. It shows that a bigger fraction of clients with more training data can lead to better model performance since more useful knowledge can be learned. According to Table D.7, compared to 100%-client-based F-TSL, 80%-client-based F-TSL offers comparable performance in terms of TP and DITA with acceptable training time. It makes 80% an appropriate setting for balancing performance and efficiency. To save space, we do not report results for the TaxiRome datasets, as these are similar to those obtained on Tdrive.

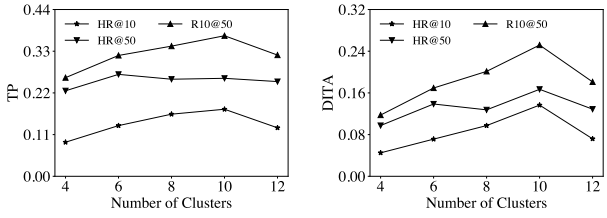


(a) TP on Tdrive (b) DITA on Tdrive

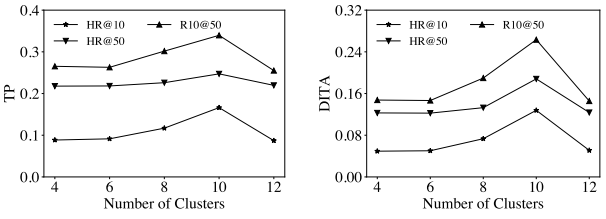


(c) LCRS on Tdrive (d) NetERP on Tdrive

Fig. D.6: Effect of Client Fractions on Tdrive



(a) TP on Tdrive (b) DITA on Tdrive



(c) TP on TaxiRome (d) DITA on TaxiRome

Fig. D.7: Effect of the Number of Clusters on Both Datasets (100 Clients)

Effect of the Number of Clusters

We next study how sensitive the F-TSL is w.r.t. the number of clusters, where we consider 100 clients and 4, 6, 8, 10, and 12 clusters. The results are given in Figure D.7. We observe that all curves first increase slightly and then drop. It is reasonable to use 10 clusters in this set of experiments. When the number of

clusters changes from 10 to 12, the model performance decreases as it is more difficult to make consistent updates between the server and clients. This leads to ineffective feature learning. To save space, we only report the results of TP and DITA on the two datasets; similar results are obtained in the evaluation of LCRS and NetERP. Also, similar results are obtained when setting the number of clients to 50 and 80.

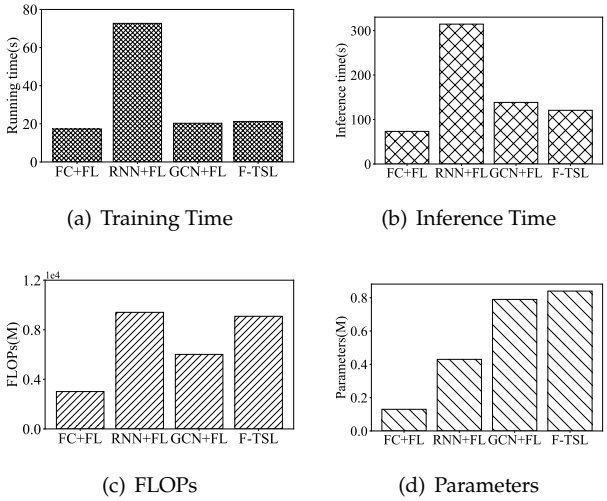
Table D.5: Centralized Method vs. F-TSL

Datasets	**Metrics**		**Method**	
			ST2Vec	F-TSL
Tdrive	LCRS	HR@10	0.1973	0.1330
		HR@50	0.1734	0.2088
		R10@50	0.2704	0.3531
	NetERP	HR@10	0.1495	0.2433
		HR@50	0.1744	0.3170
		R10@50	0.1816	0.4166
TaxiRome	LCRS	HR@10	0.0621	0.0698
		HR@50	0.1013	0.1033
		R10@50	0.1629	0.1749
	NetERP	HR@10	0.0892	0.0819
		HR@50	0.1217	0.1256
		R10@50	0.1909	0.1765

Centralized Method vs. F-TSL

To compare the proposed F-TSL with the centralized method in terms of trajectory similarity learning, we select ST2Vec [23], a recent state-of-the-art deep learning framework for centralized trajectory similarity computation, which considers fine-grained spatial and temporal relations between trajectories. To save space, we only report the results of LCRS and NetERP, in Table D.5. We can see that F-TSL generally performs worse than ST2Vec on Tdrive and TaxiRome since it is easier for the centralized method to learn commonalities from centralized data with shuffled mini-batch training. But the centralized method reduces privacy severely, which makes the learned model vulnerable. In addition, we observe that F-TSL performs comparably with ST2Vec on TaxiRome. This is because that data contained in TaxiRome is sparse, and the proposed privacy-preserving clustering mechanism in F-TSL can alleviate the influence of sparsity and achieve more effective feature learning.

To save space, we only report the results of LCRS and NetERP on the two datasets; similar results are obtained in the evaluation of TP and DITA.



(a) Training Time (b) Inference Time

(c) FLOPs (d) Parameters

Fig. D.8: Resource Efficiency Comparison

Resource Efficiency

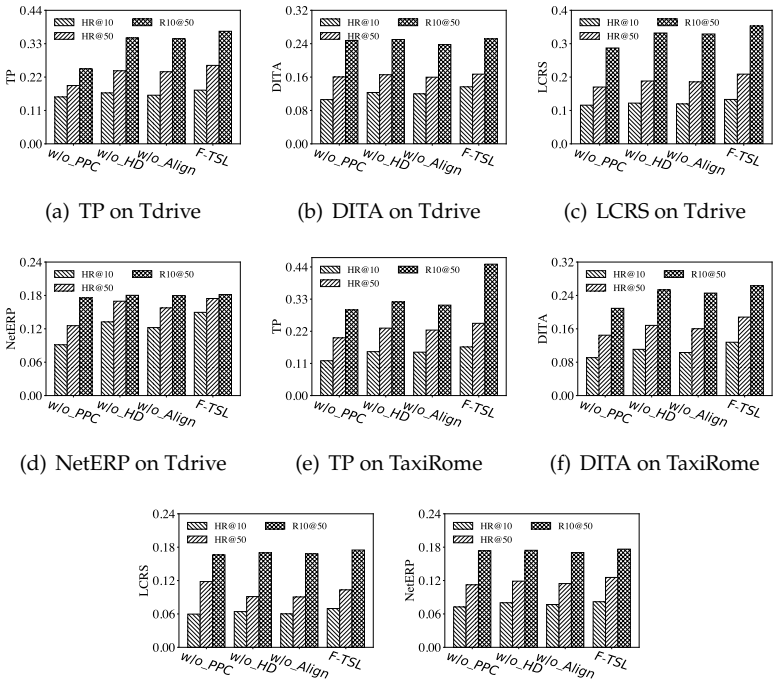
As resource efficiency is important in federated trajectory similarity computation to enable scalability, we study the training time (of each epoch), inference time (of each epoch), floating-point operations (FLOPs), and the number of parameters for all the deep-learning-based methods on the TaxiRome dataset. Figures D.8(a) and D.8(b) report the training and inference times. We see that the training time of F-TSL is lower than that of RNN+FL, which is because the proposed privacy-preserving clustering mechanism enables faster model update. In addition, F-TSL is comparable to other baselines in terms of training time. Although FC+FL takes less time for inference, it performs worse than F-TSL in terms of accuracy. This indicates the feasibility of F-TSL for model deployment in real trajectory similarity learning scenarios.

It is clear that FC+FL and GCN+FL consume fewer FLOPs and have fewer parameters than F-TSL. However, they are much less accurate and are worse than F-TSL by up to 57.94% in terms of HR@10. Moreover, F-TSL uses fewer FLOPs and has more parameters than the most accurate competing model, RNN+FL, while achieving higher accuracy. Note that FLOPs equate system overhead. Thus, F-TSL has smaller client-server communication costs between clients compared to RNN+FL, which yields better performance.

Ablation Study

To gain insight into the effects of key aspects of F-TSL, we evaluate three F-TSL variants.

- ***w/o Privacy-Preserving Clustering (w/o_PPC)***. *w/o_PPC* is the F-TSL



(a) TP on Tdrive (b) DITA on Tdrive (c) LCRS on Tdrive

(d) NetERP on Tdrive (e) TP on TaxiRome (f) DITA on TaxiRome

(g) LCRS on TaxiRome (h) NetERP on TaxiRome

Fig. D.9: Performance of F-TSL and Its Variants on Both Datasets

model without the privacy-preserving clustering mechanism.

- ***w/o Hausdorff Distance (w/o_HD)***. *w/o_HD* replaces the Hausdorff distance with l_1*-norm* distance to compute the similarity between trajectory embeddings.
- ***w/o Alignment Loss (w/o_Align)***. *w/o_Align* is the F-TSL model without the alignment loss L_{align}.

To evaluate whether the components in F-TSL all contribute to the performance of trajectory similarity learning, we compare F-TSL with its variants. The results are shown in Figure D.9. Regardless of the datasets, F-TSL always outperforms its counterparts without privacy-preserving clustering, Hausdorff distance, or alignment loss, which shows that these three components are all useful to trajectory similarity learning since removing any one of them will decrease the prediction accuracy. Further, on both datasets, *w/o_PPC* performs the worst among all variants in most cases. F-TSL outperforms *w/o_PPC*, improving the performance by up to 49.79%, 28.30%, 22.78%, and 63.74% in terms of TP, DITA, LCRS, and NetERP, respectively, which offers evidence of the effectiveness of the proposed privacy-preserving clustering mechanism.

Table D.6: Scalability: Effect of the Number of Clients on Both Datasets

Datasets	Number of Clients (s/epoch)			
	20	50	80	100
Tdrive	19.44	20.19	20.74	22.25
TaxiRome	19.52	20.30	20.88	21.16

Table D.7: Scalability: Effect of Client Fractions on Both Datasets

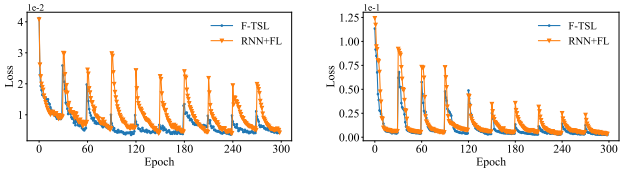
Datasets	Client Fractions (s/epoch)			
	20%	50%	80%	100%
Tdrive	21.04	21.54	25.22	28.07
TaxiRome	22.54	23.21	25.60	28.68

Scalability

To assess the scalability of F-TSL, we conduct experiments to study the effect of different numbers of clients on both datasets in terms of training time. The results are shown in Table D.6. Generally, the training time for each training epoch of F-TSL increases slightly with an increase in clients. These results indicate that F-TSL scales with the number of clients. In addition, F-TSL scales with client fractions. We study the effect of client fraction by sampling four different fractions of clients in each cluster: 20%, 50%, 80%, and 100%. Table D.7 shows the results of running time for each client including the local training time and communication time. We observe that the running time increases approximately linearly with the client fractions.

Convergence Analysis of Training

We study the training convergence of F-TSL and the best-performing baseline, RNN+FL, on the TaxiRome and Tdrive datasets — see Figure D.10. The number of epochs for global aggregation and local training are set to 10 and 30, respectively, which means that we train local data for each client with 30 epochs, then upload parameters to the central server for model aggregation, performing the entire process ten times. F-TSL converges faster than RNN+FL on both datasets, which is mainly due to the proposed privacy-preserving clustering mechanism. The privacy-preserving clustering aggregates similar clients together, which ensures consistent model updates, and thus accelerates model convergence. We also observe that the loss margin between F-TSL and RNN+FL on TaxiRome is larger than that on Tdrive. This is because data contained in TaxiRome is sparse and the proposed privacy-preserving clustering mechanism is capable of handling sparsity and learning more powerful features.



(a) TaxiRome (b) Tdrive

Fig. D.10: Training Convergence of F-TSL on Both Datasets

6 Related Work

We briefly review prior studies on trajectory similarity and federated learning, which are highly relevant to our study.

6.1 Trajectory Similarity

Trajectory similarity learning [2, 5, 7, 38, 39] has attracted substantial research interest due to the increasing availablility of trajectory data and the rich downstream applications. Traditionally, distance-based methods such as DTW [5], LCSS [9], and ERP [38] have been used for trajectory similarity computation. They first map trajectories to road network paths and then define similarity based on distance measures. For example, Koide et al. [14] propose NetERP through aggregating shortest-path distances between the vertices of two trajectories. However, these non-learning methods generally are computationally expensive to use.

To solve this problem, recent studies [7, 13, 23] apply neural networks to compute trajectory similarity by leveraging their powerful approximation capabilities. In particular, these deep learning based methods first embed the input trajectories and learn distance functions to approximate similarities. Traj2SimVec [12] is proposed to consider sub-trajectory similarity in the learning process. Further, GTS [7] is developed to target trajectory similarity learning in road networks. Most recently, Fang et al. [23] propose ST2Vec to compute trajectory similarity considering fine-grained spatio-temporal dependencies.

In addition, many studies target trajectory similarity computation for particular trajectory-related applications [40–42]. Chen et al. [41] devise index and pruning techniques to accelerate similarity search and joins over trajectory data, where tree-based index structures [40] (e.g., K-D tree or R-tree) are applied to organize trajectories. Zhang et al. [42] conduct comprehensive experiments with state-of-the-art trajectory simplification methods.

However, these methods are trained with centralized data. They disregard decentralized settings and also do not take privacy into account.

6.2 Federated Learning

Federated Learning (FL) is a machine learning approach where many clients (often called edge devices) collaboratively train a model using decentralized data under the orchestration of a central server [27]. Several studies review recent progress in FL [27, 43], which provide definitions, architectures, and applications. Generally, FL can be organized into three categories: horizontal FL, vertical FL, and federated transfer learning. Horizontal FL [28, 44] is suitable for applications where user features of datasets overlap substantially with each other but where there is little overlap among users. Vertical FL [45, 46] is useful in cases where little overlap exists among the user features of datasets, while the users overlap substantially with each other. In the case of federated transfer learning [47], there is little overlap of both user features and users. This approach is usually applied to overcome the lack of data or tags without data segments.

FL has recently been applied to spatio-temporal data (e.g., traffic data) with the aim of ensuring privacy. For instance, FRA [48] explores spatial data aggregation queries in federated scenarios. Next, FL has been applied to trajectory-related applications (such as human mobility prediction and travel mode identification) [49–52]. However, these studies are designed for trajectory classification or prediction problems, and cannot be applied to trajectory similarity computation directly. They either only consider raw GPS-based trajectories and ignore the information of road networks [49–51], which is utilized in the trajectory similarity measures, or study trajectory homomorphic encryption algorithm in FL [52]

Although existing studies provide solutions for training a model with decentralized data, they cannot be applied readily to trajectory similarity learning since they either fail to capture the inherent spatio-temporal correlations that characterize GPS-based trajectories or lack the consideration of a road network.

7 Conclusion

We propose a federated trajectory similarity learning framework entitled F-TSL for decentralized trajectory similarity computation, which is based on horizontal federated learning aiming at protecting privacy. To capture complex spatio-temporal correlations of trajectories, we design a local trajectory preprocessing and learning module with a spatio-temporal encoder and decoder for each client. A novel privacy-preserving clustering mechanism is proposed to content with trajectory heterogeneity. In addition, an alignment loss is proposed to keep consistent model updates between the server and clients. An empirical study on two real datasets demonstrates the effectiveness of the

proposed HF-STL framework in trajectory similarity learning.

References

[1] P. Zhao, A. Luo, Y. Liu, J. Xu, Z. Li, F. Zhuang, V. S. Sheng, and X. Zhou, "Where to go next: A spatio-temporal gated network for next poi recommendation," *TKDE*, vol. 34, no. 5, pp. 2512–2524, 2020.

[2] Z. Fang, S. Gong, L. Chen, J. Xu, Y. Gao, and C. S. Jensen, "Ghost: A general framework for high-performance online similarity queries over distributed trajectory streams," *SIGMOD*, vol. 1, no. 2, pp. 1–25, 2023.

[3] C. Zheng, X. Fan, C. Wang, and J. Qi, "Gman: A graph multi-attention network for traffic prediction," in *AAAI*, 2020, pp. 1234–1241.

[4] Z. Li, Y. Zhao, N. Botta, C. Ionescu, and X. Hu, "Copod: copula-based outlier detection," in *ICDM*, 2020, pp. 1118–1123.

[5] B.-K. Yi, H. V. Jagadish, and C. Faloutsos, "Efficient retrieval of similar time sequences under time warping," in *ICDE*, 1998, pp. 201–208.

[6] S. Atev, G. Miller, and N. P. Papanikolopoulos, "Clustering of vehicle trajectories," *TITS*, vol. 11, no. 3, pp. 647–657, 2010.

[7] P. Han, J. Wang, D. Yao, S. Shang, and X. Zhang, "A graph-based approach for trajectory similarity computation in spatial networks," in *SIGKDD*, 2021, pp. 556–564.

[8] Y. Chang, J. Qi, Y. Liang, and E. Tanin, "Contrastive trajectory similarity learning with dual-feature attention," in *ICDE*, 2023, pp. 2933–2945.

[9] M. Vlachos, G. Kollios, and D. Gunopulos, "Discovering similar multidimensional trajectories," in *ICDE*, 2002, pp. 673–684.

[10] Z. Shang, G. Li, and Z. Bao, "Dita: distributed in-memory trajectory analytics," in *SIGMOD*, 2018, pp. 725–740.

[11] H. Yuan and G. Li, "Distributed in-memory trajectory similarity search and join on road network," in *ICDE*, 2019, pp. 1262–1273.

[12] H. Zhang, X. Zhang, Q. Jiang, B. Zheng, Z. Sun, and W. Sun, "Trajectory similarity learning with auxiliary supervision and optimal matching.(2020)," in *IJCAI*, 2020, pp. 11–17.

[13] P. Yang, H. Wang, Y. Zhang, L. Qin, W. Zhang, and X. Lin, "T3s: effective representation learning for trajectory similarity computation," in *ICDE*, 2021, pp. 2183–2188.

[14] S. Koide, C. Xiao, and Y. Ishikawa, "Fast subtrajectory similarity search in road networks under weighted edit distance constraints," *PVLDB*, vol. 13, no. 11, pp. 2188–2201, 2020.

[15] M. Guo, X. Jin, N. Pissinou, S. Zanlongo, B. Carbunar, and S. S. Iyengar, "In-network trajectory privacy preservation," *CSUR*, vol. 48, no. 2, pp. 1–29, 2015.

[16] J. Xiong, R. Ma, L. Chen, Y. Tian, Q. Li, X. Liu, and Z. Yao, "A personalized privacy protection framework for mobile crowdsensing in iiot," *TII*, vol. 16, no. 6, pp. 4231–4241, 2019.

[17] Z. Wu, G. Li, S. Shen, X. Lian, E. Chen, and G. Xu, "Constructing dummy query sequences to protect location privacy and query privacy in location-based services," *WWWJ*, vol. 24, no. 1, pp. 25–49, 2021.

[18] Z. Huo, X. Meng, H. Hu, and Y. Huang, "You can walk alone: Trajectory privacy-preserving through significant stays protection," in *DASSFA*, 2012, pp. 351–366.

[19] C. Meng, S. Rambhatla, and Y. Liu, "Cross-node federated graph neural network for spatio-temporal data modeling," in *SIGKDD*, 2021, pp. 1202–1211.

[20] K. L. Musliner, T. Munk-Olsen, W. W. Eaton, and P. P. Zandi, "Heterogeneity in long-term trajectories of depressive symptoms: Patterns, predictors and outcomes," *Journal of affective disorders*, vol. 192, pp. 199–211, 2016.

[21] J. Taylor, X. Zhou, N. M. Rouphail, and R. J. Porter, "Method for investigating intradriver heterogeneity using vehicle trajectory data: A dynamic time warping approach," *Transportation Research Part B: Methodological*, vol. 73, pp. 59–80, 2015.

[22] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *TCC*, 2006, pp. 265–284.

[23] Z. Fang, Y. Du, X. Zhu, D. Hu, L. Chen, Y. Gao, and C. S. Jensen, "Spatio-temporal trajectory similarity learning in road networks," in *SIGKDD*, 2022, pp. 347–356.

[24] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk, "On map-matching vehicle tracking data," in *PVLDB*, 2005, pp. 853–864.

[25] S. Shang, L. Chen, Z. Wei, C. S. Jensen, K. Zheng, and P. Kalnis, "Trajectory similarity join in spatial networks," *PVLDB*, vol. 10, no. 11, pp. 1178–1189, 2017.

[26] R. S. D. Sousa, A. Boukerche, and A. A. Loureiro, "Vehicle trajectory similarity: models, methods, and applications," *CSUR*, vol. 53, no. 5, pp. 1–32, 2020.

[27] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *TIST*, vol. 10, no. 2, pp. 1–19, 2019.

[28] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *AISTATS*, 2017, pp. 1273–1282.

[29] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 24, no. 2, pp. 84–90, 1981.

[30] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. Quek, and H. V. Poor, "Federated learning with differential privacy: Algorithms and performance analysis," *TIFS*, vol. 15, pp. 3454–3469, 2020.

[31] Y. Aono, T. Hayashi, L. Wang, S. Moriai *et al.*, "Privacy-preserving deep learning via additively homomorphic encryption," *TIFS*, vol. 13, no. 5, pp. 1333–1345, 2017.

[32] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *SIGKDD*, 2016, pp. 855–864.

[33] C. Dwork, "Differential privacy," in *ICALP*, 2006, pp. 1–12.

[34] M.-F. Balcan, T. Dick, Y. Liang, W. Mou, and H. Zhang, "Differentially private clustering in high-dimensional euclidean spaces," in *ICML*, 2017, pp. 322–331.

[35] V. Cohen-Addad, A. Epasto, S. Lattanzi, V. Mirrokni, A. Munoz Medina, D. Saulpic, C. Schwiegelshohn, and S. Vassilvitskii, "Scalable differentially private clustering via hierarchically separated trees," in *SIGKDD*, 2022, pp. 221–230.

[36] J. Chen, R. Wang, L. Liu, and J. Song, "Clustering of trajectories based on hausdorff distance," in *ICECC*, 2011, pp. 1940–1944.

[37] C. T Dinh, N. Tran, and J. Nguyen, "Personalized federated learning with moreau envelopes," *NeurIPS*, vol. 33, pp. 21 394–21 405, 2020.

[38] L. Chen and R. Ng, "On the marriage of lp-norms and edit distance," in *PVLDB*, 2004, pp. 792–803.

[39] L. Deng, Y. Zhao, J. Chen, S. Liu, Y. Xia, and K. Zheng, "Learning to hash for trajectory similarity computation and search," in *ICDE*, 2024, pp. 4491–4503.

[40] Z. Chen, H. T. Shen, X. Zhou, Y. Zheng, and X. Xie, "Searching trajectories by locations: an efficiency study," in *SIGMOD*, 2010, pp. 255–266.

[41] L. Chen, S. Shang, C. S. Jensen, B. Yao, and P. Kalnis, "Parallel semantic trajectory similarity join," in *ICDE*, 2020, pp. 997–1008.

[42] D. Zhang, M. Ding, D. Yang, Y. Liu, J. Fan, and H. T. Shen, "Trajectory simplification: an experimental study and quality analysis," *PVLDB*, vol. 11, no. 9, pp. 934–946, 2018.

[43] Q. Li, Y. Diao, Q. Chen, and B. He, "Federated learning on non-iid data silos: An experimental study," in *ICDE*, 2022, pp. 965–978.

[44] J. Zhang, Y. Wu, and R. Pan, "Incentive mechanism for horizontal federated learning based on reputation and reverse auction," in *WWW*, 2021, pp. 947–956.

[45] X. Luo, Y. Wu, X. Xiao, and B. C. Ooi, "Feature inference attack on model predictions in vertical federated learning," in *ICDE*, 2021, pp. 181–192.

[46] X. Jin, P.-Y. Chen, C.-Y. Hsu, C.-M. Yu, and T. Chen, "Cafe: Catastrophic data leakage in vertical federated learning," *NeuIPS*, vol. 34, pp. 994–1006, 2021.

[47] Y. Liu, Y. Kang, C. Xing, T. Chen, and Q. Yang, "A secure federated transfer learning framework," *IEEE Intelligent Systems*, vol. 35, no. 4, pp. 70–82, 2020.

[48] Y. Shi, Y. Tong, Y. Zeng, Z. Zhou, B. Ding, and L. Chen, "Efficient approximate range aggregation over large-scale spatial data federation," *TKDE*, vol. 35, no. 1, pp. 418–430, 2021.

[49] Y. Zhu, Y. Liu, J. James, and X. Yuan, "Semi-supervised federated learning for travel mode identification from GPS trajectories," *TITS*, vol. 23, no. 3, pp. 2380–2391, 2021.

[50] J. Feng, C. Rong, F. Sun, D. Guo, and Y. Li, "PMF: A privacy-preserving human mobility prediction framework via federated learning," *IMWUT*, vol. 4, no. 1, pp. 1–21, 2020.

[51] A. Li, S. Wang, W. Li, S. Liu, and S. Zhang, "Predicting human mobility with federated learning," in *SIGSPATIAL*, 2020, pp. 441–444.

[52] M. Han, K. Xu, S. Ma, A. Li, and H. Jiang, "Federated learning-based trajectory prediction model with privacy preserving for intelligent vehicle," *International Journal of Intelligent Systems*, vol. 37, no. 12, pp. 10 861–10 879, 2022.